AN INTRODUCTION TO STM32 PROGRAMMING AND HARDWARE DESIGN

GTU Robotics Club

Preface

This book covers an introduction to STM32s hardware and programming, but its worth noting that STM32 is a very vast topic, and those all topics can’t be covered in this short “book”. I’ve provided important resources wherever necessary, do check them out.

Also, I had to write this book in a very short amount of time, there are some topics (especially in the Chapter 4) in which I’ve not explained in much detail. For those topics, I’ll create a GitHub repository and post them there.

How to use this book?

The important topics are **bolded** out.

All the online websites / resources are hyperlinked, like this one.

Code blocks are encapsulated in light blue square boxes.

```
print(“This is a code block!”)
```

Contents

1 Introduction

What is STM32? And why STM32?

STM32 is a family of microcontroller ICs based on 32-bit **RISC** (Reduced Instruction Set Computing) **ARM** Cortex-M33F, Cortex-M7F, Cortex-M4F, Cortex-M3, Cortex-M0+, and Cortex-M0 cores. They have numerous configurable options, and also STMicroelectronics (STM) attaches its own peripherals to the core before converting the design into a silicon die. In other words, they are Integrated Circuits similar to ATMEGA328p with more functionalities.

Fig. 1 STM32F407 IC

But you'll be wondering that why STM32? Is it required to use the STM32 rather than using Arduino Mega or Arduino Due?

Well in my opinion, yes STM32 is far more superior than the IC in Arduino Due (we'll be using STM32F4 series which is much more superior!).
But that's not all the benefits we will be getting by using STM32, other benefits include more number of GPIOs (we'll be using the 144 packet IC, with 114 GPIOs), customizing the board according to our needs, as we'll be using a breakout board for the ICs with which we only need to change the bottom part of the PCB as per our needs, and program the breakout board accordingly. You'll learn more about hardware design and programming further in this book, with which you'll understand why is STM32 better than other available boards out there.

But remember, STM32 has a steep learning curve, that's why most of the time, rather than learning a new board, printing a new PCB which will serve our purpose, we use already available boards, which is inexpensive and easy to use (Arduino).

Choosing a right board for yourself!

Before we start learning to program a STM32, we need to get a board, but there are various stm32 boards available out there, so which one should you buy?

But there are various types of STM32 boards, with different cores, number of pins, flash size and many more.

Before choosing a board, let's see the naming convention of a STM32 board.

We'll be using STM32F405ZGT6.

STM32	-	Family
F	-	Foundation / High Performance
4	-	Arm Cortex M4

05	-	Line
Z	-	Number of pins (144)
G	-	Flash memory size (1024Kb)
T	-	Package (LQFP)
6	-	Temperature (-40 °C to 85 °C)

You can find the information for other ICs here.

But rather than getting an MCU, and creating a breakout board, there are other development boards available out there, which will serve our purpose, which is learning in this case.

If you are low on budget, go for STM32 Blue Pill (which contains STM32F103C8) with a ST-Link v2/v3.

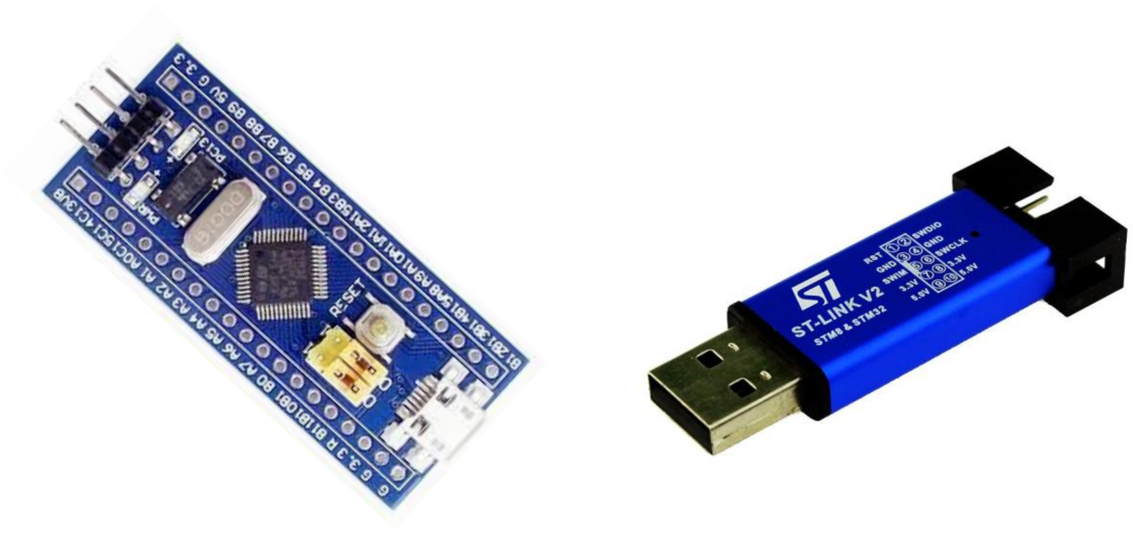


Fig. 2 STM32 Blue Pill and

ST-Link V2

The only issue of **Blue Pill** and other STM32 boards is that they **cannot be programmed directly via micro-USB cable**, and requires additional hardware connection (we need to connect 4-pins with jumper wires). But there are other development boards available out there which are shipped with ST-Link attached with them. Those boards are Discovery development board and Nucleo development board (there are other expansion boards available out there but they don't ship with ST-Link). Let me tell you the difference between these two boards, the discovery board contains some inbuilt sensors, and can be considered as an evaluation or prototyping board, whereas Nucleo board has Arduino compatible pinouts. So, for example if you have F4 and F7 Nucleo board, they can be used in the same PCB, stated that you are using Arduino compatible headers and not Morpho headers.

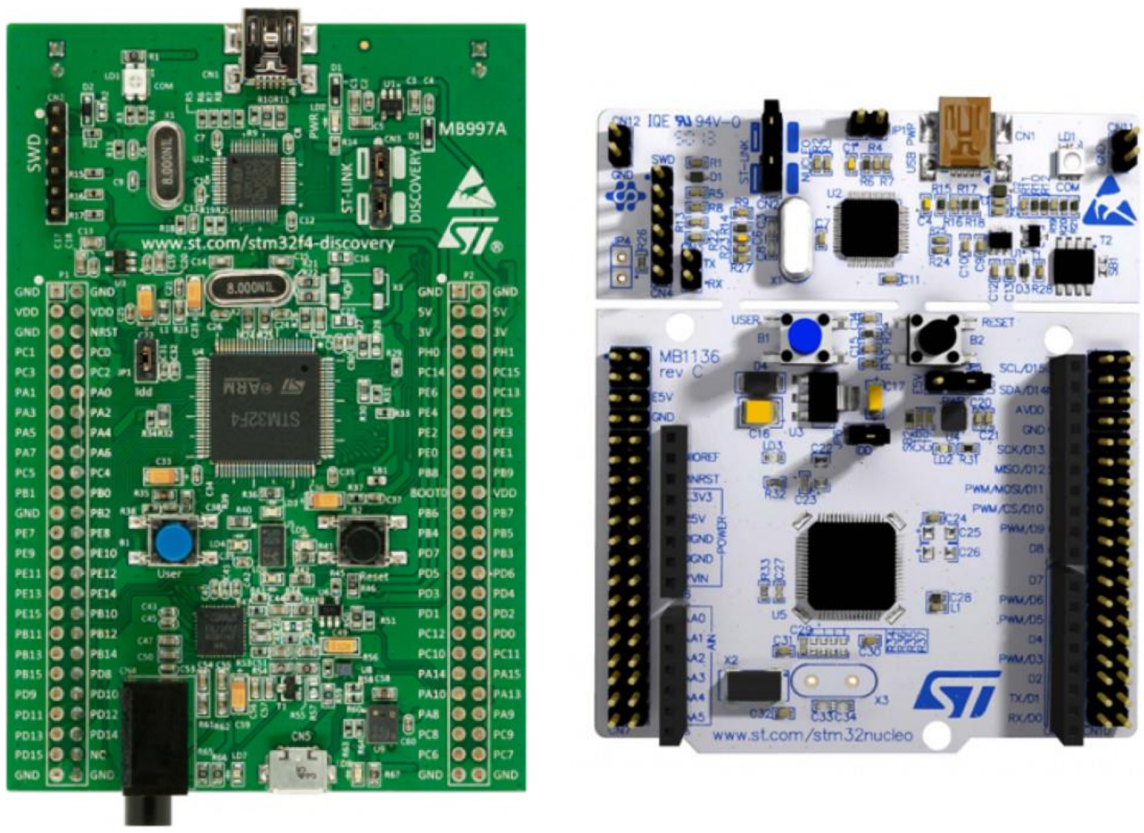


Fig. 3 STM32 Discovery board (Left)

And STM32 Nucleo board

Note that you can select any board and follow this book, I've used **STM32 Blue Pill**, **STM32F407VG Discovery Board**, and

STM32F103RB Nucleo board, only the pinouts will change and nothing else.

Different methods to program a STM32

Well, there are various ways to program a STM32.

The most basic way is to program STM32 via Arduino IDE, you'll need to configure the **Arduino IDE** to be able to program STM32 via it, but since we won't be using this method, I'll provide resources in the appendix section for the same. The advantages of this method are that you don't need to learn any additional programming (though you'll need to learn configurating the pins).

Another method is to program STM32 using online compiler i.e., **MBED**, but this method has certain drawbacks, one of them is speed is slow, and will depend on the internet connectivity, other one is not all boards are supported as MBED was essentially created for Nucleo boards. Resources for this method will also be covered in the Appendix section of this book.

We can also program STM32 via **register level programming**, the only drawback of this method is programming via register requires lots of knowledge regarding MCU, internal structure, and you should be a good programmer. Also, code for one MCU will change from others as the internal structure is not the same, we won't be covering this method, but you can always visit YouTube for the same.

The method which we'll be following throughout the book is using **HAL** i.e., **Hardware Abstraction Layer**, and programming in **STM32CubeIDE** (note that CubeMx / STM32CubeIDE can also be used, if you wish to use Keil, or other IDE).

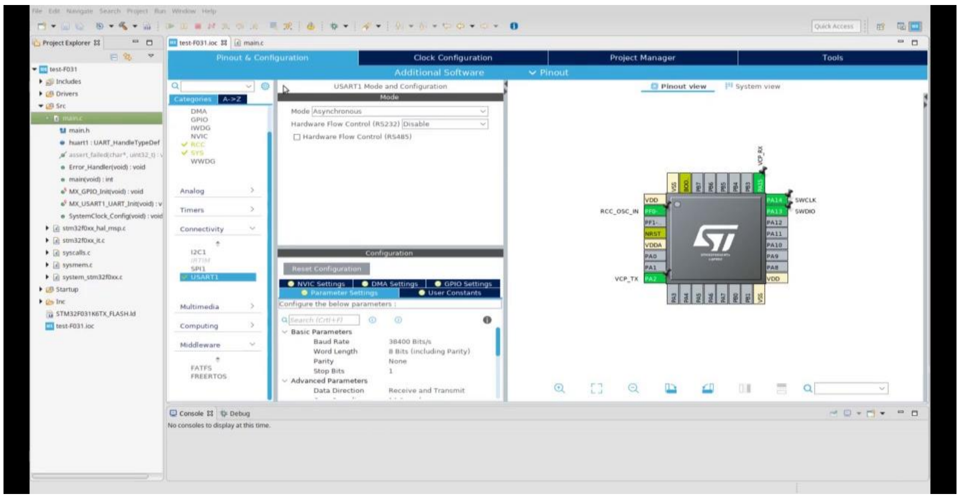
But what is Hardware Abstraction Layer? It is basically a layer of programming that allows a computer OS to interact with a hardware device (it is same as OS kernels).

STM32 HAL or STM32 abstraction layer embedded software ensures maximized portability across STM32 portfolio, and HAL APIs are available for all peripherals.

If you've not yet understood the concept of HAL, let me explain it to you in a simpler way. HAL is basically a layer which allows you to access / change the registers at a lower level. For example, you are using a Operating System, and you wish to shut down the computer, but you just have to press a button, rest all the things will be handled by the OS kernels which will ensure safely shutting down the computer by closing all the programs in the memory, which you don't need to perform explicitly. Similarly, if you want to blink a LED, rather than going to register levels, and changing the state of the GPIO, we just need to write

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, GPIO_PIN_SET);
```

(Don't get overwhelmed by the above line of code, we'll go deeper in further chapters of this book).



Mode Asynchronous
Hardware Flow Control (RS232) Disable
Basic Parameters
Baud Rate 38400 Bits/s
Word Length 8 Bits (including Parity)
Parity None
Stop Bits 1
Advanced Parameters
Data Direction Receive and Transmit
No consoles to display at this time.

Fig. 4 STM32CubeIDE

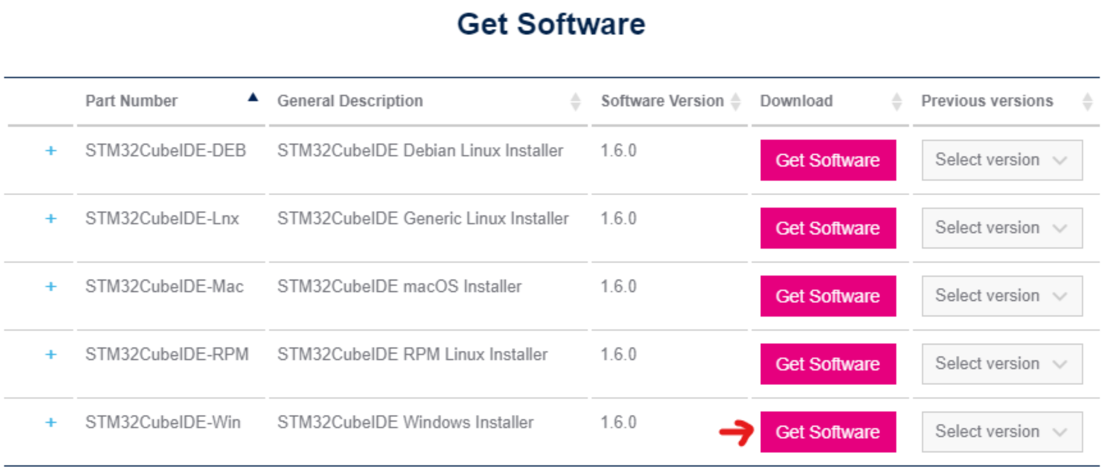
In the above picture, you can see an MCU, where we can configure the pins in a Graphical User Interface (GUI), configure the clock and many more things, once we are done with configuring the GPIOs, we only need to press a button, which will automatically generate the code for us, and we only need to add the logic (we'll see that in further section of this book). Though there are lots of advantages of using STM32CubeIDE and HAL, there are some disadvantages as well. Performance issues will be there, being an Integrated Development Environment (IDE), it is heavy on operating system, though HAL has lots of features, the documentation is not that detailed (STM is working on it), also STM32 HAL had some bugs. Even though STM32CubeIDE and HAL have some disadvantages, it is perfect for learning and makes a lot of things easier.

Installing the required software

We'll be using Windows 10 through out this book, and the list of software we need to install include STM32CubeIDE and a C compiler (we will be installing GCC for learning C Programming).

i. Installing STM32CubeIDE

Go to https://www.st.com/en/development-tools/stm32cubeide.html and click on *"Get Software"* for the OS you'll be using, in my case it's Windows 10, so I'll be downloading STM32CubeIDE-Win

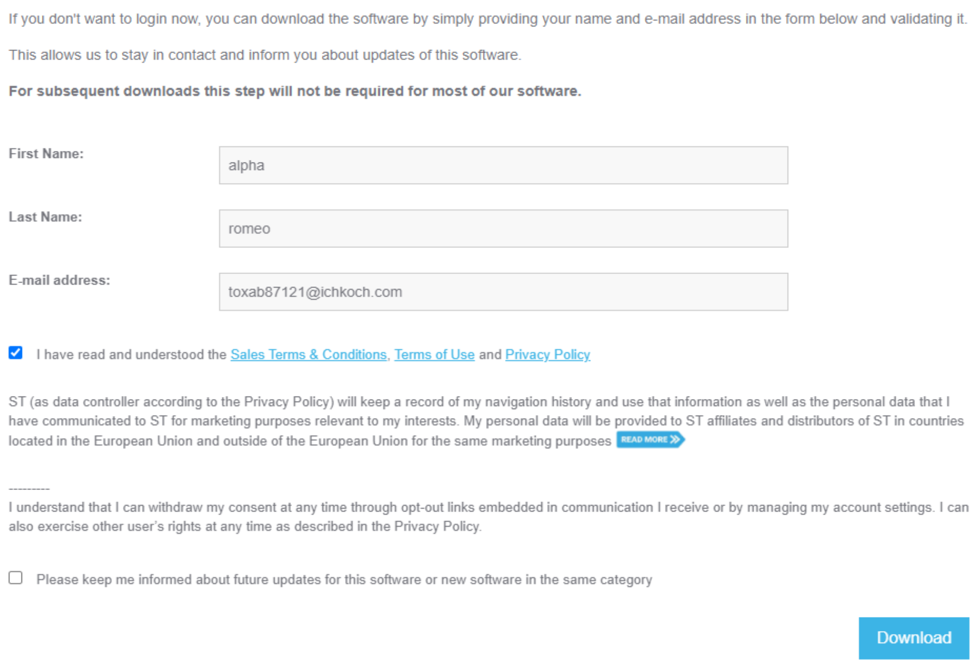


Get Software

	Part Number	General Description	Software Version	Download	Previous versions
+	STM32CubeIDE-DEB	STM32CubeIDE Debian Linux Installer	1.6.0	Get Software	Select version
+	STM32CubeIDE-Lnx	STM32CubeIDE Generic Linux Installer	1.6.0	Get Software	Select version
+	STM32CubeIDE-Mac	STM32CubeIDE macOS Installer	1.6.0	Get Software	Select version
+	STM32CubeIDE-RPM	STM32CubeIDE RPM Linux Installer	1.6.0	Get Software	Select version
+	STM32CubeIDE-Win	STM32CubeIDE Windows Installer	1.6.0	Get Software	Select version

Fig 5. Downloading STM32CubeIDE

Read and accept the *"License Agreement"*, submit your details in the form (it isn't necessary to provide your original details but make sure, you can receive emails on the address, so that you can still receive the download link).



If you don't want to login now, you can download the software by simply providing your name and e-mail address in the form below and validating it.

This allows us to stay in contact and inform you about updates of this software.

For subsequent downloads this step will not be required for most of our software.

First Name:	alpha
Last Name:	romeo
E-mail address:	toxab87121@ichkoch.com

[✓] I have read and understood the Sales Terms & Conditions, Terms of Use and Privacy Policy

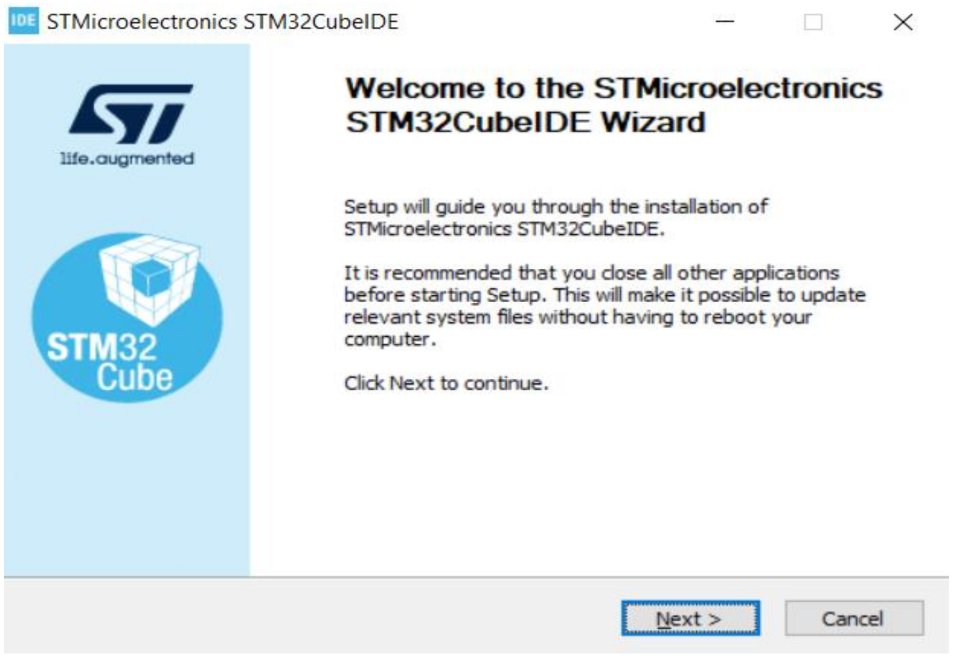
ST (as data controller according to the Privacy Policy) will keep a record of my navigation history and use that information as well as the personal data that I have communicated to ST for marketing purposes relevant to my interests. My personal data will be provided to ST affiliates and distributors of ST in countries located in the European Union and outside of the European Union for the same marketing purposes READ MORE

I understand that I can withdraw my consent at any time through opt-out links embedded in communication I receive or by managing my account settings. I can also exercise other user's rights at any time as described in the Privacy Policy.

[ ] Please keep me informed about future updates for this software or new software in the same category

Fig. 6 Filling up the details

Once you have received the email, there will be a download link / button, just click on it and grab a coffee! We'll meet once you've downloaded the STM32CubeIDE.



Welcome to the STMicroelectronics STM32CubeIDE Wizard

Setup will guide you through the installation of STMicroelectronics STM32CubeIDE.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

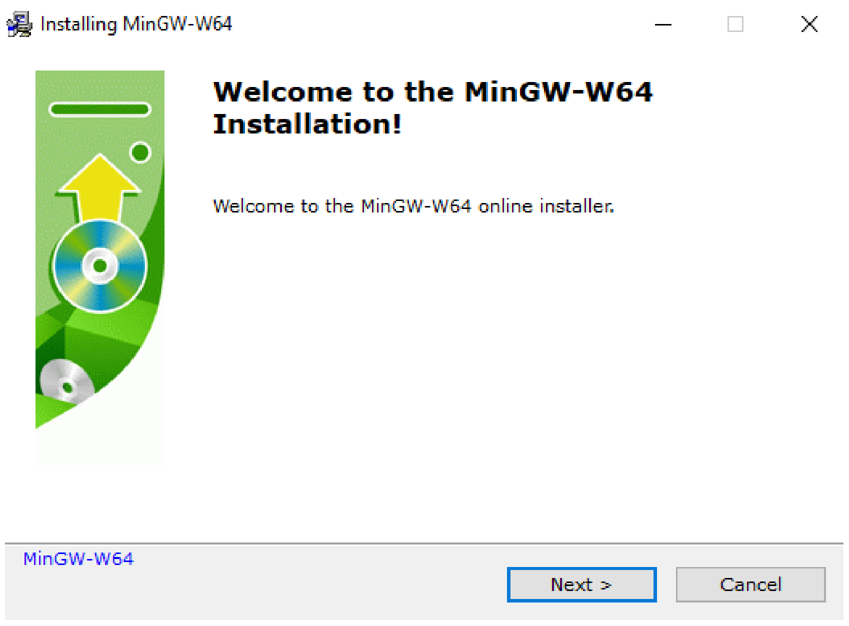
Fig. 7 STM32CubeIDE Installation

Just click next, accept the license, you'll get a window where it will ask you to install STM32 device drivers, just select all the checkboxes and proceed with the installation.

Once you've downloaded the STM32CubeIDE, we'll move further to install GCC. Note that GCC is not explicitly required for STM32CubeIDE.

ii. Installing GCC

Go to http://mingw-w64.org/doku.php/download, and select a Build compatible with your OS, we'll be selecting MingW-W64-builds. By clicking on *"MingW-W64-builds"*, we will be redirected to another page, where we will get an installation link, which will further redirect us to Source Forge, and download the installer.


Installing MinGW-W64

Welcome to the MinGW-W64 Installation!

Welcome to the MinGW-W64 online installer.

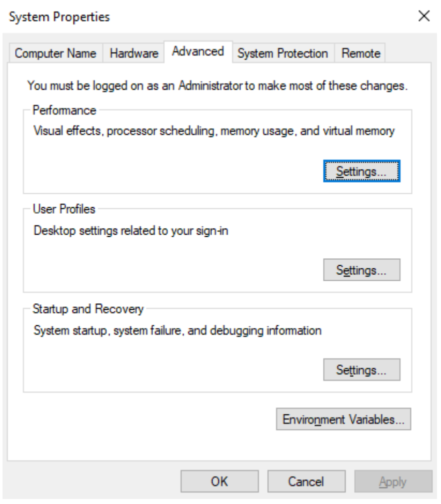
MinGW-W64

Fig. 8 MinGW-W64 Installer

Proceed with the default installation, and once finished, we need to add GCC compiler to the **Environment Variables**.

iii. Adding GCC to Environment Variables

Go to Start, and search for "Environment Variables", and click on "Edit environment variables for your account".


System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

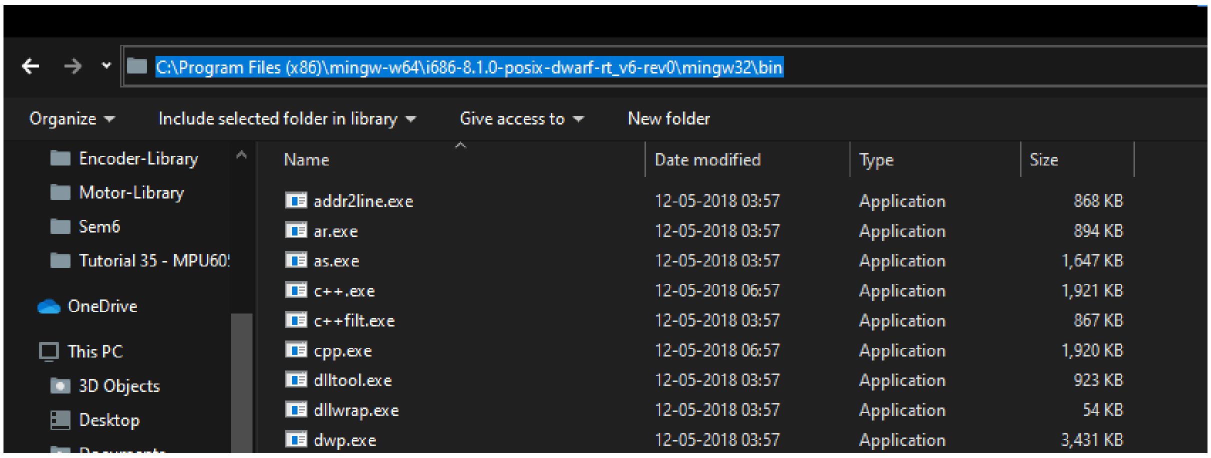
Startup and Recovery

System startup, system failure, and debugging information

Fig. 9 Environment Variables Menu

Click on "Environment Variables", double click on "Path" in User Variables. Before adding the path, we need to look for the binary folders path in our installation directory.

For getting the *path*, go to the installation directory, by default it will be *"C:\Program Files (x86)\mingw-w64\"*. Further search for bin, by opening the folder with name starting with *i686** (note that * is any string after i686), look for bin inside the mingw32 folder.



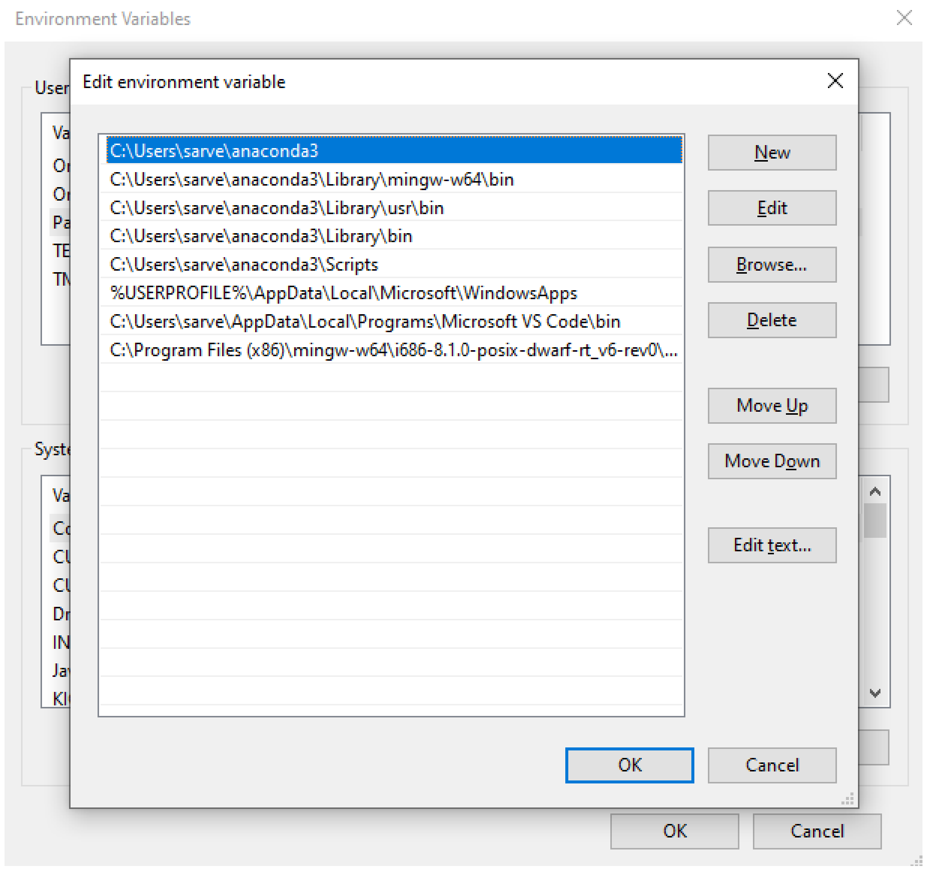
C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin

Name	Date modified	Type	Size
addr2line.exe	12-05-2018 03:57	Application	868 KB
ar.exe	12-05-2018 03:57	Application	894 KB
as.exe	12-05-2018 03:57	Application	1,647 KB
c++.exe	12-05-2018 06:57	Application	1,921 KB
c++filt.exe	12-05-2018 03:57	Application	867 KB
cpp.exe	12-05-2018 06:57	Application	1,920 KB
dlltool.exe	12-05-2018 03:57	Application	923 KB
dllwrap.exe	12-05-2018 03:57	Application	54 KB
dwp.exe	12-05-2018 03:57	Application	3,431 KB

Fig. 9 Copying the path

Once copied the path, (which is *"C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin"* in our case) paste it by clicking new in *"Edit Environment variable"* tab.

Click on OK, and add the variable.

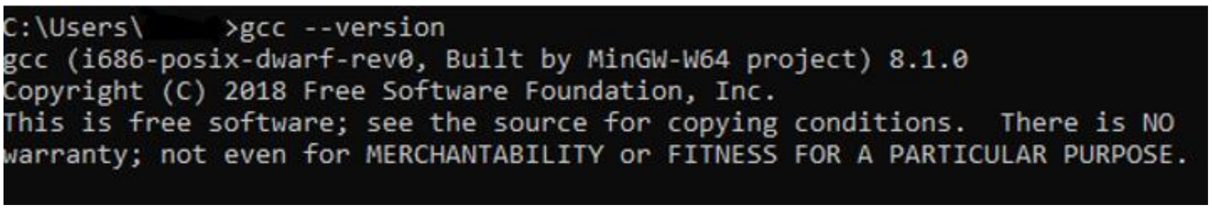


```
C:\Users\sarve\anaconda3
C:\Users\sarve\anaconda3\Library\mingw-w64\bin
C:\Users\sarve\anaconda3\Library\usr\bin
C:\Users\sarve\anaconda3\Library\bin
C:\Users\sarve\anaconda3\Scripts
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\sarve\AppData\Local\Programs\Microsoft VS Code\bin
C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\...
```

Fig. 10 Setting the variable

Let's verify that GCC is installed successfully on our computer.

Open the Windows *"Command Prompt"* (this can be done by searching cmd in start menu, or pressing Win+R to open the run window, and type "cmd" in text field, and click on OK)



```
C:\Users\[redacted]>gcc --version
gcc (i686-posix-dwarf-rev0, Built by MinGW-W64 project) 8.1.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Fig. 11 Verifying the installation

Once the command prompt is open, type gcc –version if you get a version number (something similar to Fig. 11), then congratulations! You have successfully installed GCC on your Windows OS. If you get some error like "GCC is not recognized…" means you've missed a step. Don't worry, if you are having trouble installing GCC on your Windows machine, you can follow this tutorial on YouTube or you can even use Linux OS, as they have GCC already installed.

That's all the software we'll be requiring throughout this book. See you in the next chapter!

2 Getting Started with Programming

C Programming

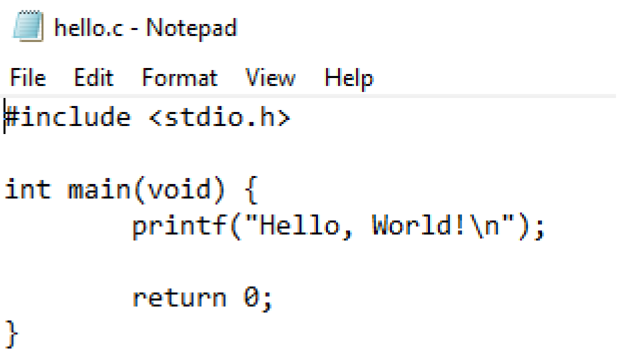
By this time, you'll be excited to program your STM32 board! But wait, before we start, we need to go through some important concepts of C Programming. Note that I won't be covering this topic in detail (if you really interested to learn C Programming in detail, I'd recommend you O'Reilly's Head First C book, or Harvard's CS50 Introduction to Computer Science on edX) as there are already lot of books and resources available.

Let's start with the famous "Hello, World!" program, and we'll break it down to understand the structure of C Program. In STM32CubeIDE, we follow the same structure while programming.

```
#include <stdio.h>

int main(void) {

  printf("Hello, World!\n");

  return 0;
}
```

Before we move on to the details of the program, let’s compile it and see the output.

How to compile this file? It’s very simple, just create a file with *“.c”* (I’ve saved it as hello.c) extension and paste the above code snippet in it.



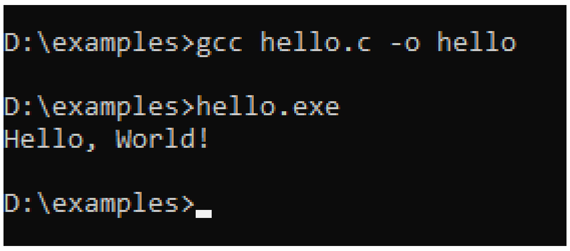
```c
#include <stdio.h>

int main(void) {
        printf("Hello, World!\n");

        return 0;
}
```

Fig. 12 hello.c program

In order to compile the code, we need to use GCC (i.e., GNU Compiler Collection). Open the command prompt in the directory where you created *“hello.c”* program (you can do this by **shift+right-click** in the folder, and clicking on *“Open PowerShell window here”* and use the following command to compile the *“.c”* file.

```
gcc hello.c -o hello
```

In the above command we are passing hello.c to gcc, and -o flag symbolizes the name of the output file. In this case, the output file will be named as a.exe (or a.o in Linux OS). If we don’t use the -o flag, we’ll be getting compiled file as a.exe (or a.out in Linux OS). Once you execute the command, and if your code is error free, you’ll not get any output, if you get any output, it can be either warnings or errors. Make sure you solve them before executing the program. Once compiled properly, we can proceed on executing the program, this can be done by simply typing hello.exe and pressing enter in command prompt (or we’ll need to change the permissions in Linux OS, by sudo chmod +x hello.o and run by ./hello.o).



```
D:\examples>gcc hello.c -o hello

D:\examples>hello.exe
Hello, World!

D:\examples>_
```

Fig. 13 Execution of hello.exe

Let’s move on to detailed explanation of the above code.

```
#include <stdio.h>
```

Here, stdio.h is a header file, we are telling the compiler to include / copy-paste the functions which are present in *“stdio.h”*, so that we can use it in our code. stdio.h is for standard input and output. Here we are using *“printf”* function, which requires this library to be included. Compiler searches for function’s parameter and return value, which can be found in the same.

```
int main(void)
```

This is simply a function with name as *“main”*, which returns *“int”* and takes no argument (which is symbolized by void). Note that every *“.c”* file should contain a function named *“main”* or it can be considered as an entry point for execution of the code.

```
printf(“Hello, World!\n”);
```

This line simply prints "Hello, World!" to the stdout (standard output), and \n is the new line **escape character**. (While we are at escape characters, there are various other escape characters, check out this article for more information).

```
return 0;
```

By returning a value, you can check whether the execution of the code was successful or not. If our code returns 0, means there were no issues in the execution of the program, but if it returns a non-zero number, that means there were some issues (this needs to be set from programming side and if we use void main, we don't need to return anything).

Before we move further, here are some links which you should go through if this is the first time you are learning C:

Variables in C

C++ Compiling Process

C Macros

Input/Output

Conditional Statements and Loops

Most of the time, we need to get work done based on some condition, or we need to continuously repeat a block of code, until the condition is met, this is where **Conditional Statements and Loops** come.

In our day-to-day life, we often need to take a decision based on certain parameters, for example, you are planning to make a pizza tonight! But wait, you don't have cheese to make it… So here, your chances of eating the pizza will depend on the chances of getting cheese.

The above scenario can be represented as:

```
if(cheese == true) {
  make_pizza = true;
} else {
  make_pizza = false;
}
```

But wait, we're not going to give up! Let's go outside, and buy some cheese for our pizza. We need to get a total of 1kg of cheese (just an example). We visit to one of the supermarkets, and have managed to get around 200g of cheese. But we need to buy 800g more of cheese. This scenario can be represented as a loop. Until the total weight of cheese is 1Kg, buy more cheese!

In C, we can use if statements, if-else statements, if-else-if statements, switch case, ternary operators.

And various loops control like while, do-while, and for loops. The syntax of conditionals statements and loops is given below.

Conditional Statements

Loops

There are certain things we should keep in mind while using a loop in our code. Note that, until and unless some task needs to be done infinitely (like void loop() in Arduino), the loop should break whenever the condition is satisfied. In the above example, once we get 1 or more than 1kg of cheese, we'll stop buying, think if we keep on buying more and more cheese for our whole life. When some variable's value needs to be calculated, before checking the condition, do-while loop does the job. For example, we need to calculate temperature every time in a loop, so initially calculating it before the loop, and calculating in the loop, makes the code repetitive, which is not a good practice, so we can use do-while instead.

Practice time

1. Write a program to reverse a number of any length.
2. Write a program to check if the number is palindrome or not.

Functions in C

There are certain times when we are repeatedly doing a task. For example, we want to reverse 10 numbers, so writing the same code for reversing a number again and again doesn't makes sense, right? This is why we use functions is C. Remember "int main(void)" from our hello.c program, it was a function, which returned an integer and didn't took any arguments.

There can be multiple types of function.

- No return value and no argument passed
- Return value and no arguments passed
- No return value and arguments passed
- Return value and arguments passed

It basically depends on our use, if we want to reverse a number, we'll need to pass the number as an argument and return the reversed number to the main function, but if we simply need to print, we don't need to return a value.

Let's write a program, add.c which will accept two numbers as argument, and return the addition of those two numbers.

The program is written in the next page, what we are doing here is in the second line we are declaring the function, so that compiler knows in advance that there is a function called add, which accepts 2 arguments and returns an integer. If we don't declare the function, the compiler will get confused at line number 6 (int sum = add(a, b);). We could have defined the function there only, but it is a better practice to define all the functions other than main at the end. Imagine if we had hundreds of functions in one file (not a good practice), and if you defined them at the top, it would be too difficult for us to find and make changes in main function if we have to.

```
#include <stdio.h>


int add(int num1, int num2);


int main(void) {
	int a = 10;
	int b = 20;
	int sum = add(a, b);
	printf("%d\n", sum);
}


int add(int num1, int num2) {
	return num1 + num2;
}
```

On the line 6, we are calling the function add(), with arguments a and b, which further returns a + b i.e., 30, and stores it into sum. And finally, we print sum to check whether our function worked or not.

Practice time

3. Write a program to reverse a number of any length with reverse function that accepts number as a parameter and returns the reversed number.

Pass by value and pass by reference.

Before moving to Pass-by-value and Pass-by-reference, let's discuss an important concept i.e., pointers and its importance in our programming. Whenever we create a variable its value is stored in a memory location which further has a value. Pointer generally stores the address of another variable. So why to learn pointers? For example, we are working on with a variable, and we need to do operations directly on that variable by a function, rather than returning a value, we'll need to use pointer for the same. In other words, we can directly pass the address of that variable directly to the function so that the function can do the necessary changes. This concept will be useful in STM32. For example, we've a library, which will change the state of a GPIO, we'll need to pass the address of the GPIO pin to the library.

Before moving to pointer, try to create a program, which accepts a variable and increments its value, without returning it. You may notice that the value doesn't change in the main function as the function when passed with a variable, creates local variable which is only accessible by the function. So, what we do instead is pass the address of the variable, and our main function will accept a pointer. Note that, to access the value of the pointer, we need to further refer it via pointer.

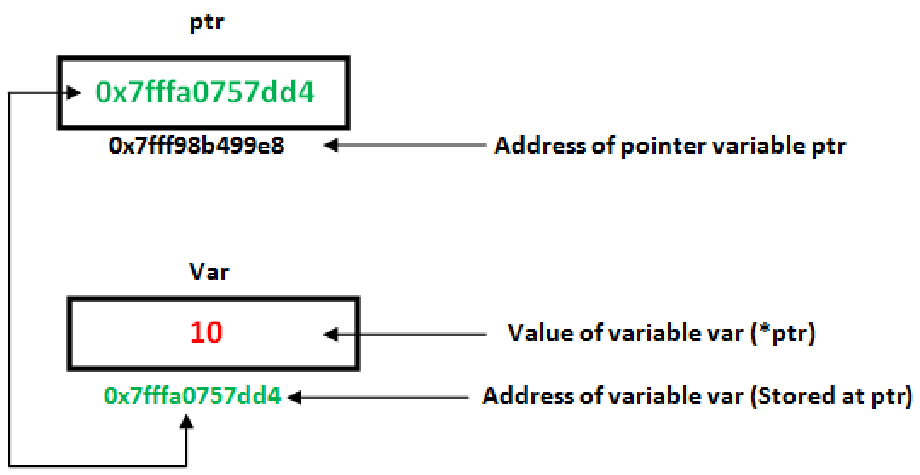

ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (Stored at ptr)

Fig. 14 Pointers in C

To create a pointer in C, we’ll use “*” operator before the variable name, and to pass the address of the variable, or simply see the address of the variable, we’ll use the “&” operator before the variable name.

```
#include <stdio.h>


void increment(int *a) {
        *a += 1;
}


int main(void) {


        int a = 0;
        increment(&a);
        printf("%d", a);



        return 0;
}
```

Go through these resources, before we move to STM32's programming!

Pointers in C

Arrays in C

Structure in C

There is lot more in C programming, but we'll be leaving it here only, as the required things have been covered / or the resource are provided.

Some things you need to keep in mind while programming is that all the codes / programs you write should work in all case scenarios. For say, you are writing a program to reverse a number, it should reverse a number of any length, i.e., it shouldn't be valid for only certain test cases. Your program should be definitive, that means it should not run forever and do the required task. And at the end remember, *"Programming is Art"*.

Hello STM32 and GPIO Programming

In this section we'll be learning the basic GPIO programming and finally blinking a LED on our STM32F103C8 (STM32 Blue Pill).

Now it's time to launch STM32CubeIDE, and do the necessary pinout configuration. Click on the File -> New - > STM32 Project. Now in the MPU/MCU selector, select the board you'll be using, I'll be using STM32F103C8. Select the MCU from the list and in the next tab finally name your project. Note that if this is the first time you are using STM32CubeIDE / MCU (Different series), it'll download some necessary files.

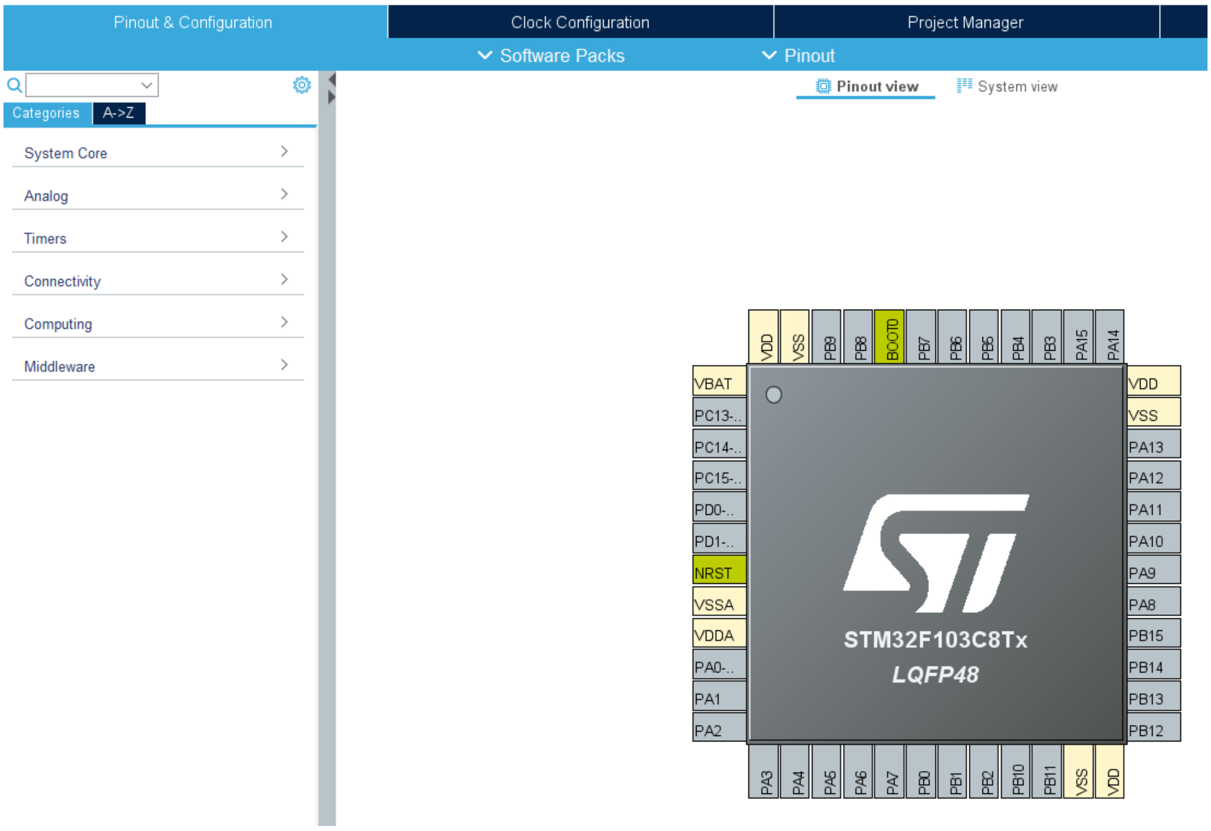


Fig. 15 New Project

We’ll be basically working in Project Configuration and Clock Configuration. Select on PC13 and set it to GPIO_Output (as it is the default LED in STM32 Blue Pill).
Then click on “System Core” tab and move to “SYS” tab, and change **No Debug** to **Serial Wire**.

In **RCC** tab in “System Core”, change HSE from Disable to Crystal / Ceramic Resonator.

Move to “Clock Configuration” tab, and change the PLL source to “HSE”, and in “System Clock Mux” use PLLCLK. Make sure there is no “x” / red errors in Clock configuration, if there exists an error, switch to that field, and press enter, that will automatically do the necessary configurations.

Once we are done with all these configurations save the project, and you will be prompted to generate the code (if you do any changes in .ioc file, you’ll be asked for code generation).

Now you’ll be automatically switched to a different perspective, and the “main.c” file would’ve been opened.

Now, let’s try to switch on the built-in LED. We’ll be writing that code in **/* USER CODE BEGIN 3 */** tab. Note that, try to write code in these comments, if you write any code / line out of these comments, whenever you do any configurations in the pins and regenerate the code, those lines would be deleted.

There are two pin states, i.e., High and Low, and we can change this state by HAL_GPIO_WritePin(PortNumber, PinNumber, State) function, and it accepts 3 parameters. We can set a pin HIGH by “SET” parameter or “GPIO_STATE_SET”, and to set a GPIO low, the same can be done with “RESET” or “GPIO_STATE_RESET” parameter.

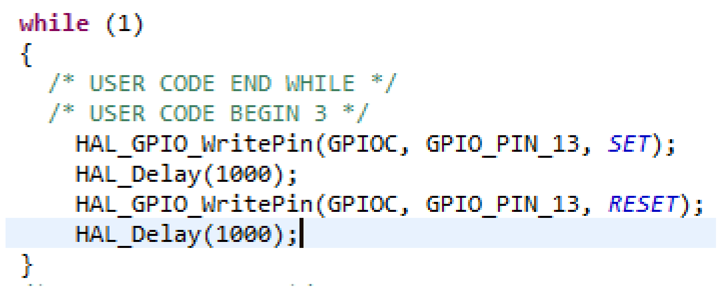
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, SET);

The above line of code will set the PC13 pin to HIGH (this function accepts 3 parameters, first is the port, next GPIO pin, and the state of pin which we wish to set).

Delay can be added by HAL_Delay(uint8_t 32); and we’ve to pass the delay in milliseconds. And to get the number of milliseconds elapsed since last boot or reset, we can use the “HAL_GetTick()” function or timers which we will be seeing in further section of this book.

So, let’s try to blink the LED using delay.

Refer Fig. 16 for the code.



```
while (1)
{
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, RESET);
    HAL_Delay(1000);
}
```

Fig. 16 LED Blink Code

Note that we can also use HAL_GPIO_TogglePin(Port Number, Pin Number); which will change the previous state of the GPIO pin, you can research more about this function on your own.

Before we run this program, we need to connect our STM32 Blue Pill with ST-Link, which can be done as shown in Fig. 17.

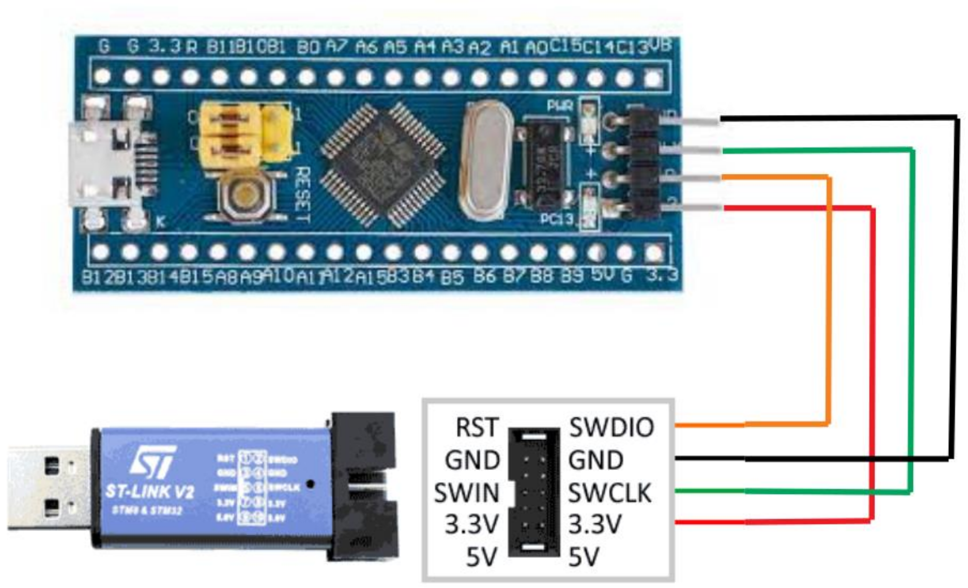


Fig. 17 Connecting ST-Link with Blue Pill

Once we have written the code, let’s run it. To-do so, simply press on the green play button ▶ or go-to run menu and run as STM32 Cortex-M C/C++ Application.

Now for example, we need to get the analog values from a sensor, say potentiometer, what we can do is configure a pin via Analog tab. There are two ADC (Analog to Digital) channels in STM32F103C8 (can vary with different MCU).

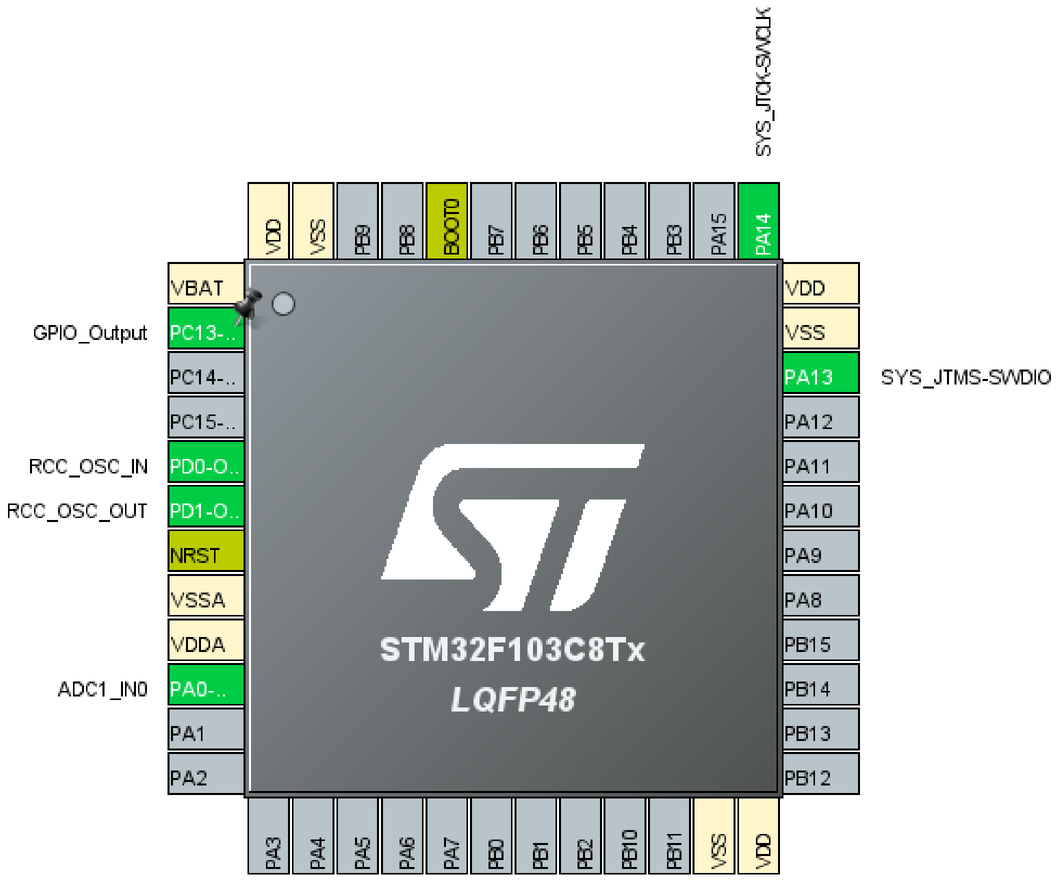
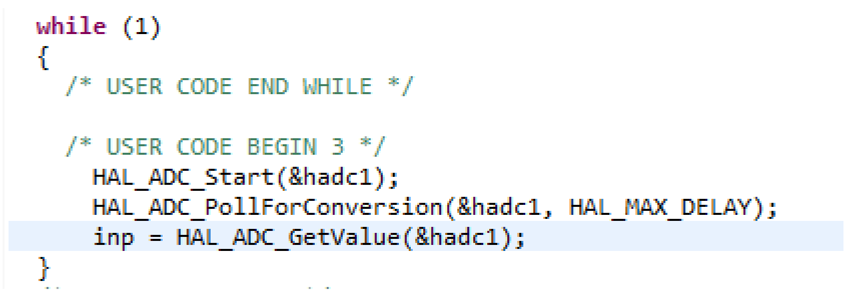


Fig. 18 Pin configuration for ADC

Once the pin is configured, we can generate the code, and start the ADC in main.c file by "HAL_ADC_Start(&hadc1); (We are passing the address of the hadc variable, so that function can directly make changes to the variable). Next, we need to get the reading via
HAL_ADC_PollForConversion(address of ADC, and a delay),
and lastly, we can store the value in a variable (typically an unsigned variable) by
HAL_ADC_GetValue(address of adc);



```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    inp = HAL_ADC_GetValue(&hadc1);
}
```

Fig. 19 Code for reading from ADC
(and inp is of datatype uint16_t)

But you'll be wondering how to output this data on a serial monitor? There are several ways to do so!

Let's discuss this in the next section, i.e., Debugging your STM32!

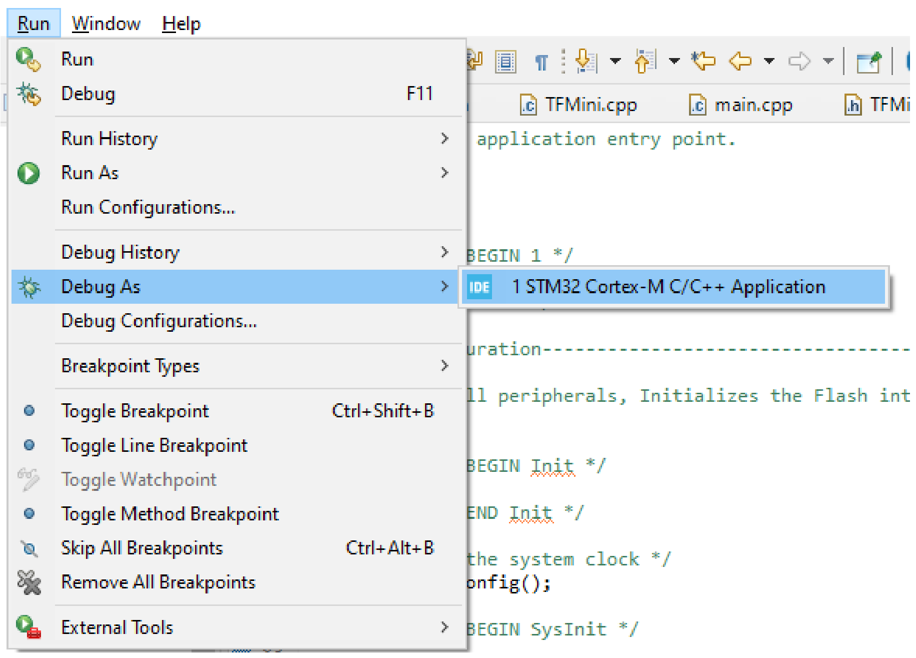
Before we move further, here's a cool feature about STM32CubeIDE, that whenever you need to auto-complete a line, you can do so by pressing ctrl+space. Also, you can right click on a function and open its declaration and view the function in depth.

Debugging your STM32

One of the easiest way to debug in STM32 is using the debugger in STM32CubeIDE. Let’s create a program, in which we’ll create a simple variable in /* USER CODE BEGIN 1 */ and in the while loop we’ll increment the variable every one second. Let’s see the different ways to track the value of that variable, starting off with the debugger in STM32CubeIDE.

Create a new project, generate the code after doing the necessary configurations, and create a variable before HAL_Init() line, i.e., in /* USER CODE BEGIN 1 */, and increment the variable in /* USER CODE BEGIN 3 */, and add delay for 1 second.

Now, go to Run Tab, Debug As, and select STM32 Core…



```
application entry point.
BEGIN 1 */
uration-----------------------------------
ll peripherals, Initializes the Flash int
BEGIN Init */
END Init */
the system clock */
onfig();
BEGIN SysInit */
```

Fig. 20 Debugging in STM32CubeIde

A menu will be prompted regarding debug configurations, proceed with default configurations, switch to debug perspective, and follow the below given steps.

Go to Window->Show View, and click on expressions. Then double click on the variable whose value you need to trace, and right click on it, and click on “Add to Watch Expression…”

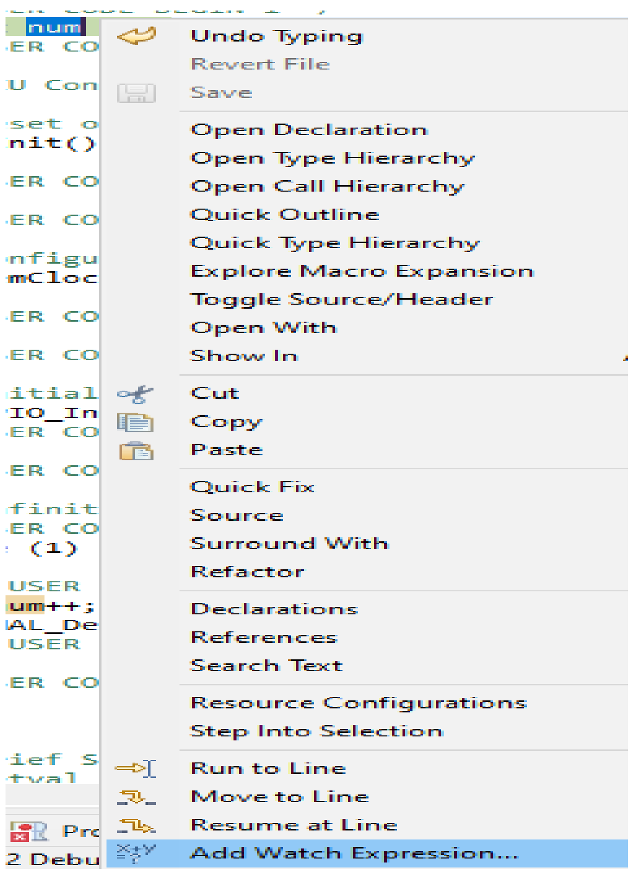


Fig. 21 Adding variable to watch expression

Lastly, we’ll add a break point, where the variable increments, this can be done by right clicking at that line number, and clicking on “Toggle Breakpoint”.

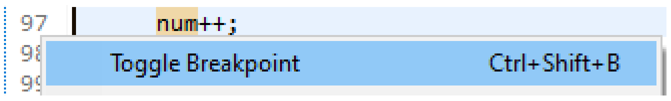
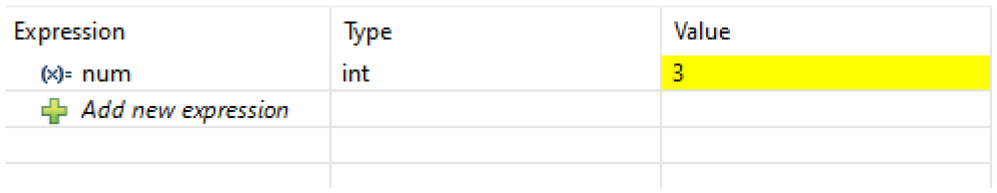

```
num++;
```

Fig. 22 Toggling a breakpoint.

Now, we’ll need to resume the execution of our program, by clicking on resume button (F8 is the shortcut key), you’ll notice that the execution of the program stops, whenever we hit the breakpoint, again resume the code, and you'll notice the value increases in Expressions tab.



Expression	Type	Value
(x)= num	int	3
Add new expression		

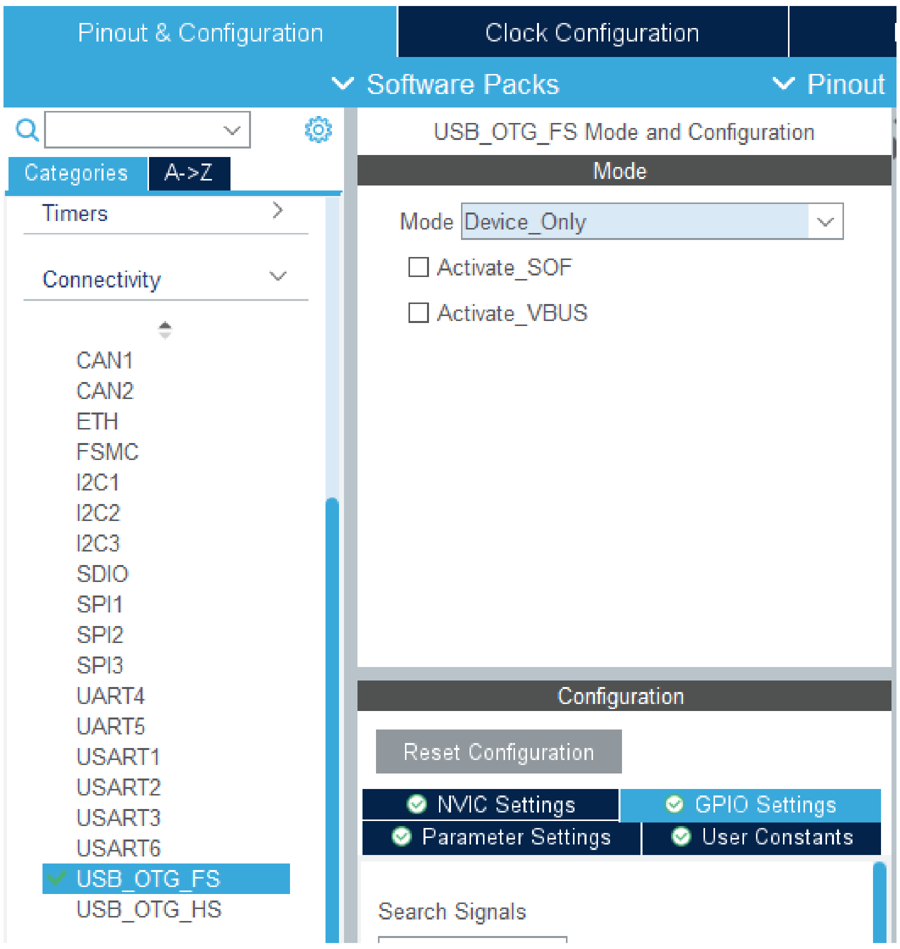
Fig. 23 Variable's value increments

Next method is to get the data by serial printing. There are further two ways to do so, but since the other method is only compatible with Nucleo boards, as it's UART2 pins are connected with ST-Link, which enables Serial data transfer via UART, we won't be covering this method here, but I've created a library for the same, and the same can be accessed from here.

Another method is to create a Virtual Com Port (VCP), on STM32 Blue Pill (I've used a discovery board, but you can follow the same for STM32 Blue Pill, but note that check the datasheet if you wish to use another board / MCU, as all the MCU don't support this method.

Create a new project and do the pin configurations as given below.

In the USB_OTG_FS section of Connectivity tab, set the mode from Disable to Device_Only, and in the USB_Device tab of the Middleware tab, set the mode to Communication Device Class for "Class for FS IP". Refer fig. 24 and fig. 25 for the same.

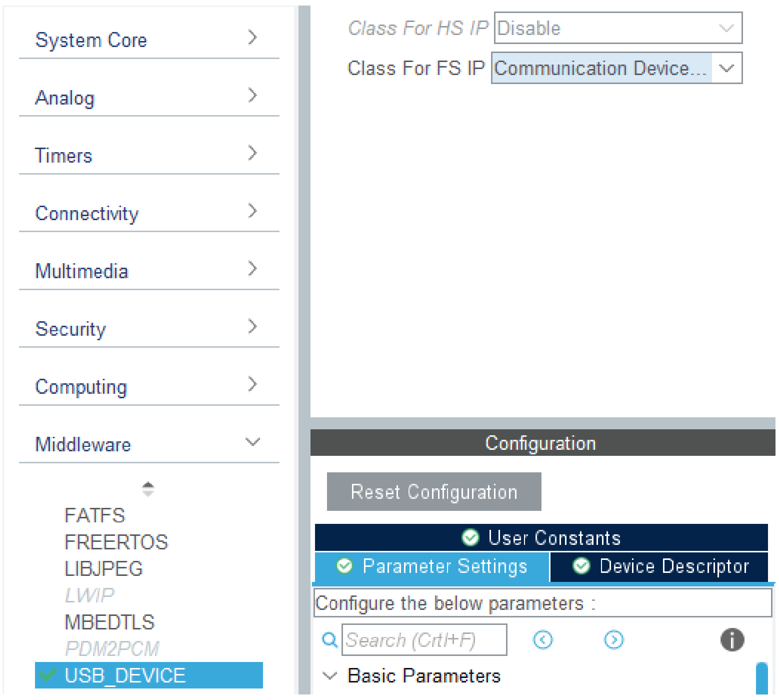


Mode Device_Only

☐ Activate_SOF

☐ Activate_VBUS

Fig. 24 Configuring Connectivity for VCP



Class For HS IP Disable

Class For FS IP Communication Device...

Configure the below parameters :

Fig. 25 Configuring Middleware for VCP

Now save the file and generate the code. Here, in order to transmit data, we need to include a library named *"usbd_cdc_if.h"* (do this in private include section of main.c file), also include the string.h and stdio.h header files.

Moving further, in main function of the code, create a char array of size 32. In your while loop, copy the string "Hello, World!\r\n" in buf using sprint(), and finally transmit the buf via USB VCP (something like serial.print). This can be done by following line of code:

```
CDC_Transmit_FS((uint8_t *)buf, strlen(buf));
```

Where buf is the name of the variable that I have created. Since CDC_Transmit_FS accepts uint8_t, we are typecasting a char to uint8_t and passing the pointer of the same (you can learn more about typecasting here). Let's run the program. Now in order to output the data on the screen, let's open a command shell console. Follow the below mentioned steps to do so.

Click on the Open Console (this icon), and further click on command shell console.

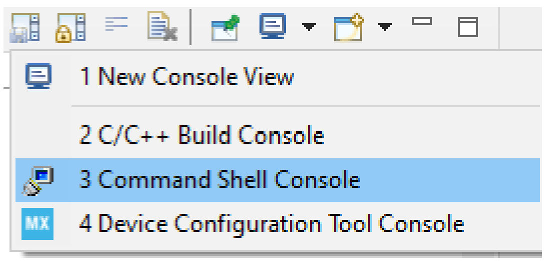
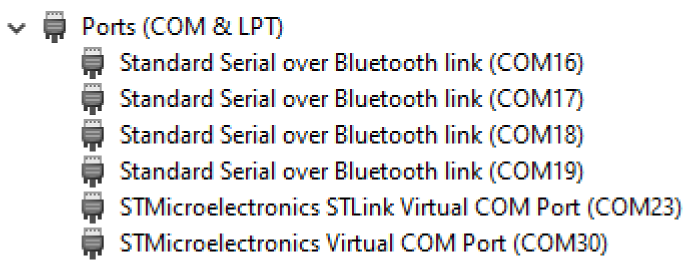


Fig. 24 New Command Shell Console

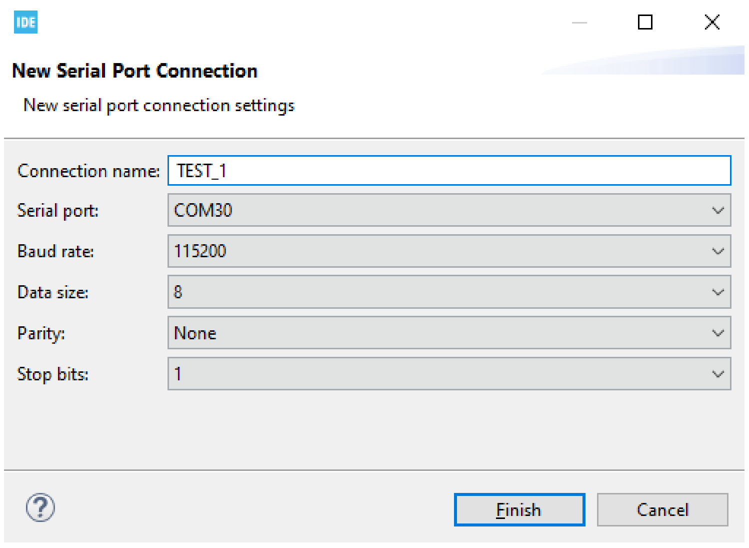
You'll need to create a new connection name, for the same open device manager (search for device manager in start, or type “*devmgmt.msc*” in run), and search for various COM Ports available.



Ports (COM & LPT)
Standard Serial over Bluetooth link (COM16)
Standard Serial over Bluetooth link (COM17)
Standard Serial over Bluetooth link (COM18)
Standard Serial over Bluetooth link (COM19)
STMicroelectronics STLink Virtual COM Port (COM23)
STMicroelectronics Virtual COM Port (COM30)

Fig. 25 Available COM ports

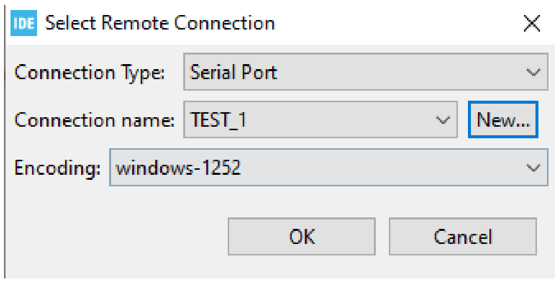
For me it’s COM30, so I’ll create connection name with COM30 in Serial Port, and will give a desired name to the Connection Name, lastly, click on Finish.



New Serial Port Connection
New serial port connection settings

Connection name: TEST_1
Serial port: COM30
Baud rate: 115200
Data size: 8
Parity: None
Stop bits: 1

Fig. 26 New Connection



Select Remote Connection

Connection Type: Serial Port
Connection name: TEST_1
Encoding: windows-1252

Fig. 27 Finalization

Finally, click on OK, and voila, you can see “Hello, World!” being printed on the Console.

Note that this method will be used mostly while debugging, as our custom hardware will be equipped with micro-USB / mini-USB ports, with which we’ll be able to create Virtual Com Port, and debug our programs.

PWM Generation, Timers and Interrupts

We've learnt a lot till now! Now we'll be learning about PWM generation in STM32, which will help us to drive motors at different speed. Create a new project, and now, go to the Timers tab, and select any timer (TIM1, TIM2 …) and set any channel to PWM Generation Channel, here I'll be using Timer 4 (TIM4) and channel 3, as it has an inbuilt LED, with which we can visualize the changes in the brightness. AFAIK, the inbuilt LED in the Blue Pill doesn't have a timer, so you can attach a LED at that specific GPIO.

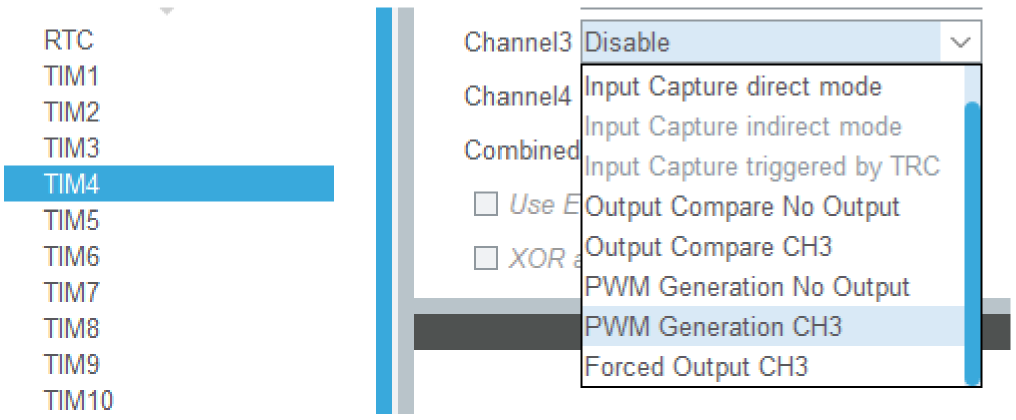
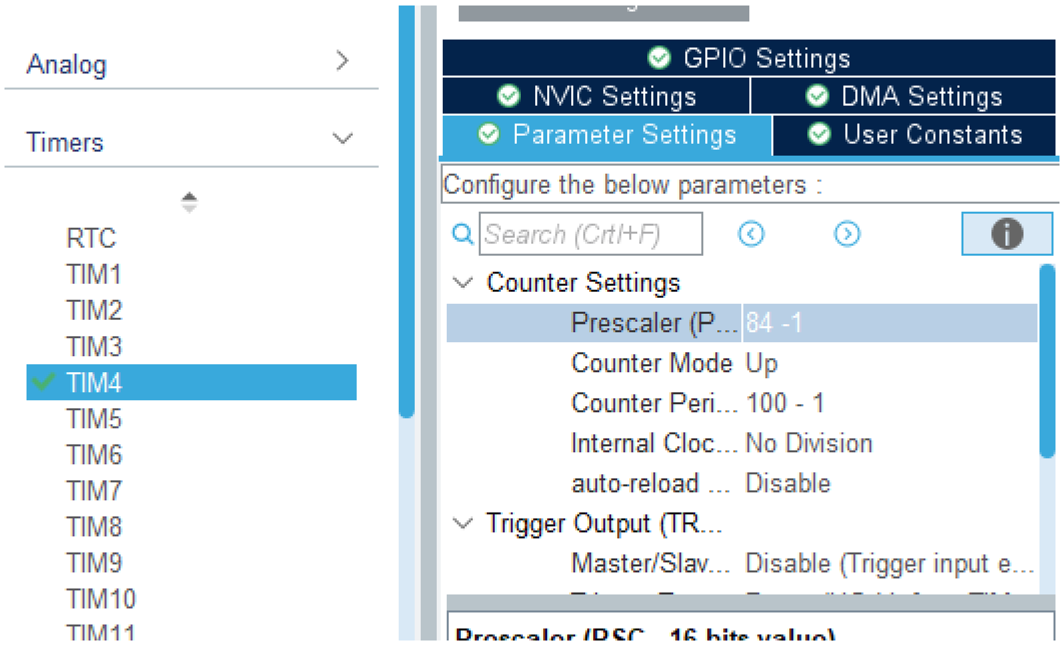


Fig. 28 Configuring PWM Generation Pin

Also configure HSE (High Speed External) clocks, and set the Debug to Serial Wire. Before generating the code, we'll have to configure some additional things as well. Firstly, we need to move to the Clock Configuration tab. Now if there are any conflicts, let the IDE solve it for you.

You'll have to refer the datasheet, that which family your TIM belongs, for me, TIM4 belongs to APB1 Family. And APB1 Timer Clock is set to 84MHz in the clock configuration. So, let's set the Prescaler in Pinout and Configuration tab, and set the counter period to desired value (I'm setting it to 100 – 1 as the counting starts from 0).



Configure the below parameters :

Counter Settings

Prescaler (P...	84 -1
Counter Mode	Up
Counter Peri...	100 - 1
Internal Cloc...	No Division
auto-reload ...	Disable

Trigger Output (TR...

Master/Slav...	Disable (Trigger input e...

Fig. 29 Configuring the Prescaler and Counter Period

But what this all means? And why are we specifically setting the Prescaler to 84 and not any other number? Since the APB1 is set to 84MHz, we'll set the frequency of the PWM generation at 1MHz, so setting the Prescaler at 84 (here 83, as counting starts form 0), will divide the APB1 with 84 for this channel, and will output 1MHz. And when we set the Counter Period to 100 (here 100 – 1), means that, one cycle will be of 100µs, and if we set the pulse to be 30 (or PWM), it will set that particular pin HIGH for 30µs and for the rest of the time (70µs) the PIN will be in RESET (LOW) state. Note that hardware PWM are always preferred over software as CPU is not wasting instructions for toggling that GPIO.

Once we are done with all this, let’s generate the code. Now after that in the /* USER CODE BEGIN 2 */ we’ll need to initialize the PWM, by the following line of code

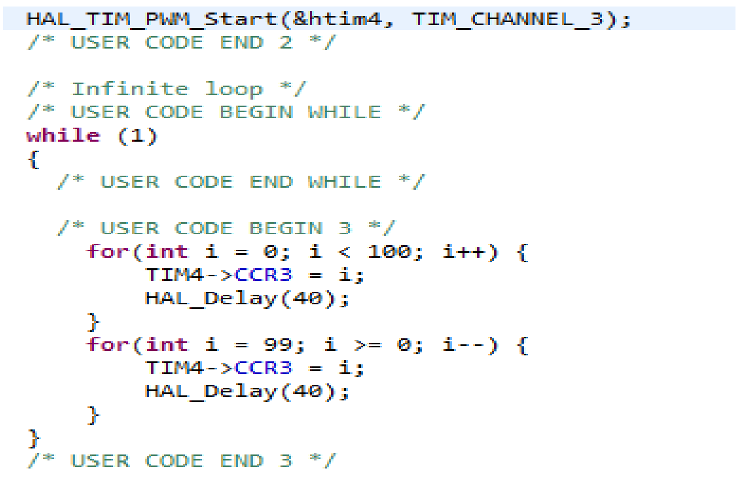
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);

where, htim4 is the 4th timer, and the second parameter is the channel, it’ll vary from different Timers and different channels.

Now in order to change the PWM (initially it’ll be zero), we need to change the CCR’s (capture/compare register) value, which is a member of a structure of type TIM_Typedef. Here we’ll be changing the value of TIM4’s CCR3 (3rd channel). This can be done by following line

TIM4->CCR3 = 50; // This will set the PWM to 50.

Below is the program to slowly increase the PWM to max and then decrease it to zero.



```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    for(int i = 0; i < 100; i++) {
        TIM4->CCR3 = i;
        HAL_Delay(40);
    }
    for(int i = 99; i >= 0; i--) {
        TIM4->CCR3 = i;
        HAL_Delay(40);
    }
}
/* USER CODE END 3 */
```

Fig. 30 Code for PWM Generation

Try playing with the Prescaler, and see the difference. Moving further let's create a program to blink a LED using timers. Create a new project, and I'll do the necessary configuration like serial wire debug and enabling HSE Crystal Oscillator. Now, let's activate a timer, which is of no use, like which doesn't have all the four channels, here I'm again using a discovery board, and will be using TIM14, which is also part of APB1.Now we want to change the Prescaler, as we want to measure time in seconds, note that Prescaler is only a 16-bit number and we can't exceed it more than 16535 (16536 – 1), so we'll be setting the Prescaler to 8400 – 1 as we need to measure time in seconds, so setting it to 8400, it will set the frequency to 10KHz (84 / 8400 = 10KHz), which will increment the counter every 0.0001s (T = 1/f, and SI unit for f is Hz, so 10KHz is 10000Hz). And lastly, we'll configure a GPIO to GPIO_Output with LED on it, and generate the code.

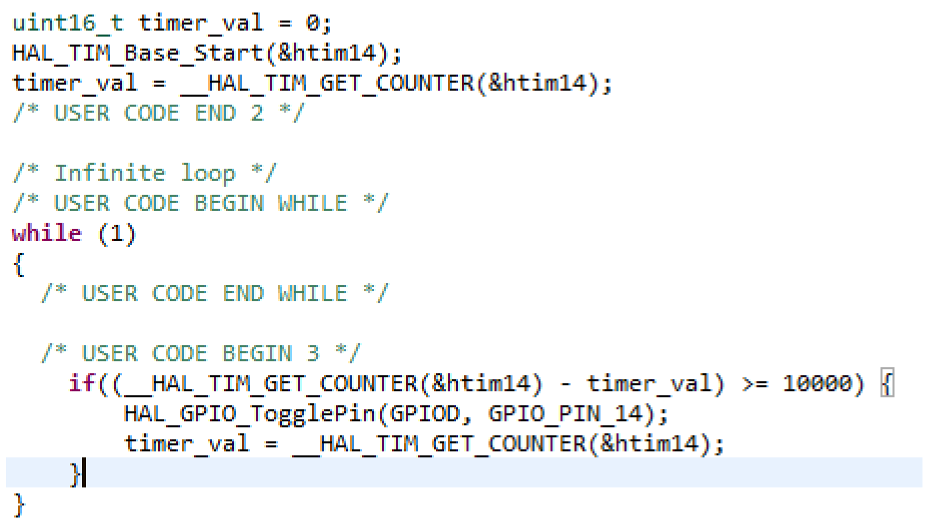
Now, let's start the timer in /* User Code Begin 2*/ by the following Line of Code (LOC)

HAL_TIM_Base_Start(&htim14); //htim14 will vary with // the timer you are using

and let's store the value of the timer immediately in a variable of datatype unsigned int of 16 bits (uint16_t) by the following Line of Code

uint16_t timer_val = __HAL_TIM_GET_COUNTER(&htim14);

and finally, refer Fig. 31 for the LED blink code.

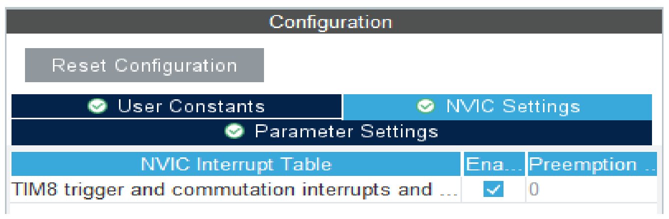


```
uint16_t timer_val = 0;
HAL_TIM_Base_Start(&htim14);
timer_val = __HAL_TIM_GET_COUNTER(&htim14);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    if((__HAL_TIM_GET_COUNTER(&htim14) - timer_val) >= 10000) {
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
        timer_val = __HAL_TIM_GET_COUNTER(&htim14);
    }
}
```

Fig. 31 Blinking LED with Timer

In the if conditional statement, what we are doing is, checking if the difference of current counter value and previous updated value is greater than 10,000 as the value of timer increments once every 0.0001s, so if we need to blink the LED once every second, the difference should be 10000. And finally, we simply toggle the LED, and update the timer value.

Lastly, let us blink this same LED with interrupts. Keep the settings same, we just need to change the value of counter period and set it to 10000 (10000 – 1), so that it'll increment once every 0.0001s, so every one second the counter will reach to 10000, and reset again. Finally, we need to enable the interrupt on this pin, the same can be done in the NVIC settings.



NVIC Interrupt Table	Ena...	Preemption ...
TIM8 trigger and commutation interrupts and ...	✓	0

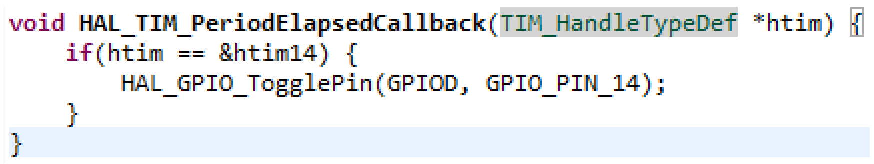
Fig. 32 Enabling interrupt

Now generate the code, and we need to start the TIM with interrupt, this can be done by

HAL_TIM_Base_Start_IT(&htim14);

where IT is interrupt. Next, we'll need to create a function in /* USER CODE BEGIN 4 */ and the function name would be HAL_TIM_PeriodElapsedCallback which returns a void, and accepts a pointer to TIM_HandleTypeDef.

Below is the image of the function created.



```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if(htim == &htim14) {
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
    }
}
```

Fig. 33 Interrupt function

So, what is happening here is our header file i.e., stm32Fxxx_hal_tim.h calls this function, whenever an interrupt occurs, and in that we are specifying if that interrupt is from htim14, then toggle the state of LED, so whenever an interrupt occurs that function is called by that header file, with the timer as an argument.

We'll be wrapping up this chapter here, there are lot's of online resources available, if you still have any doubt you can contact me.

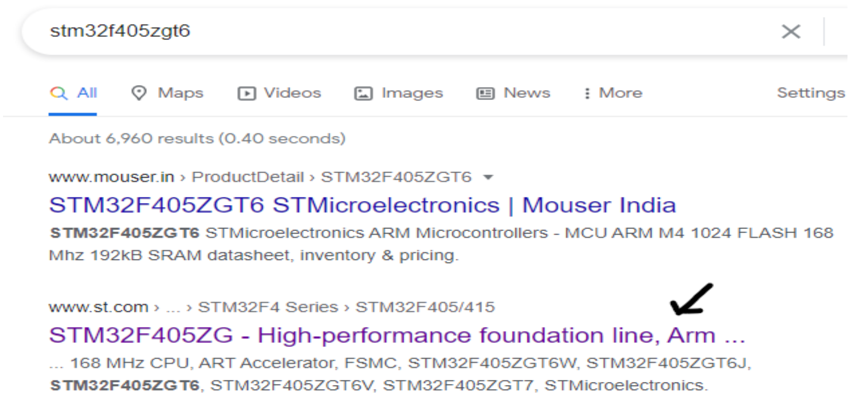
The last timers and interrupt part of this book was taken from this great YouTube tutorial.

3 Custom STM32 Hardware

In this chapter you will be learning about creating a STM32's custom hardware, and how to build a STM32 breakout from scratch. We'll look at the most important things which you shouldn't overlook while creating a breakout or custom hardware, which will serve your purpose.

As mentioned earlier that we'll be using a STM32F405ZGT6 MCU.

Before we start with the schematic, let's gather important resources. Let's google for "STM32F405ZGT6 datasheet", and visit the link with domain st.com (Refer Fig. 34).



About 6,960 results (0.40 seconds)

www.mouser.in › ProductDetail › STM32F405ZGT6
STM32F405ZGT6 STMicroelectronics | Mouser India
STM32F405ZGT6 STMicroelectronics ARM Microcontrollers - MCU ARM M4 1024 FLASH 168 Mhz 192kB SRAM datasheet, inventory & pricing.

www.st.com › ... › STM32F4 Series › STM32F405/415
STM32F405ZG - High-performance foundation line, Arm ...
... 168 MHz CPU, ART Accelerator, FSMC, STM32F405ZGT6W, STM32F405ZGT6J, **STM32F405ZGT6**, STM32F405ZGT6V, STM32F405ZGT7, STMicroelectronics.

Fig. 34 searching for datasheet.

Switch over to Documentation tab, and download the following files, ANXXXX Getting started with STM32F4xxxx MCU hardware development, ANXXXX USB hardware and PCB guidelines using STM32 MCUs and most important the Datasheet. Before moving further, let's open STM32CubeIDE, and configure the required pinouts for the breakout. Create a new project, and in MCU/MPU selector search for STM32F405ZG in part number. Give a desired name to the project, and let's configure the MCU according to our needs.

Being a breakout board, we'll be only configuring certain pins like USB, Serial Wire debug, crystal oscillator and a built-in LED.

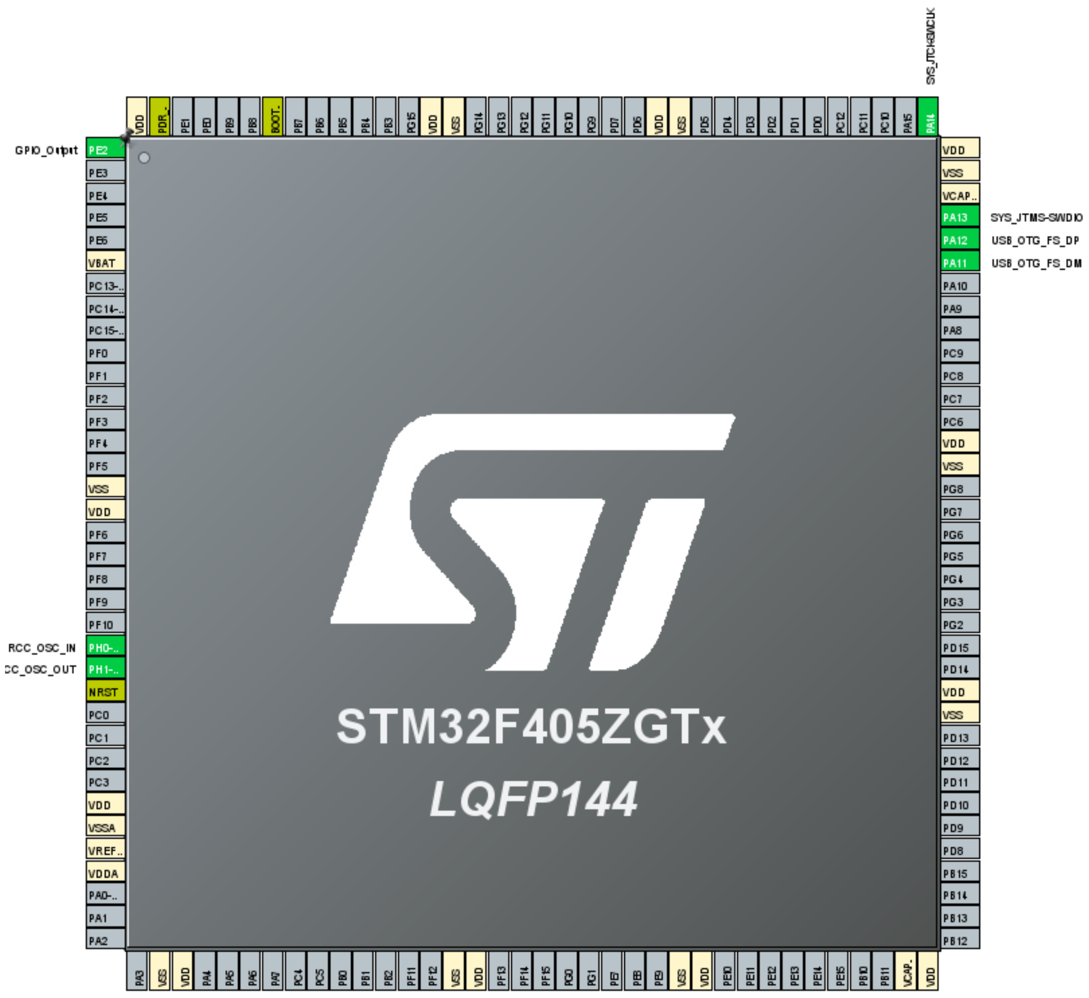


Fig. 35 Configuration of pins

In RCC tab in System Core, enable the High-Speed capacitor, and select crystal/ceramic Resonator, and after that move on to SYS tab and select Serial Wire debug. In connectivity menu, switch to USB_OTG_FS tab and from disabled select Mode as Device_Only. And last but not the least configure any pin as GPIO_OUTPUT (Just make sure that the pin doesn't include any timers or communication protocol), in this case we'll be using PE2.

Now, we've configured all the required pins, now let's move on to the schematic part of this MCU. I've already created an EasyEDA project, and already added a MCU from the library. Let's connect all Vdd with 3v3 Net Port and all the Vss with GND. Now we need to add the high-speed external crystal oscillator, which will be of 26MHz. Note that capacitance value matters a lot with crystal oscillator, as we need to add 2 capacitors, we'll need to calculate its value.

We'll be using this crystal oscillator for example, and its Load Capacitance (C_L) is 20pF.

The formula to calculate the value of both the capacitors i.e. C_1 and C_2 is given below.

$$C_L = (C_1 * C_2) / C_1 + C_2) + C_S$$

Where C_S is the Stray Capacitance. There's some stray capacitance in our PCB design, but it's value can only be guesstimated, and is usually around 2-10pF.

Now we know both the values (we are assuming C_s to be 5pF), and let's assume that C_1 == C_2, as we need to add both the capacitors of same capacitance.

We get C_1 and C_2 as 30pF. So, let's make the schematic according to the datasheet of this MCU.

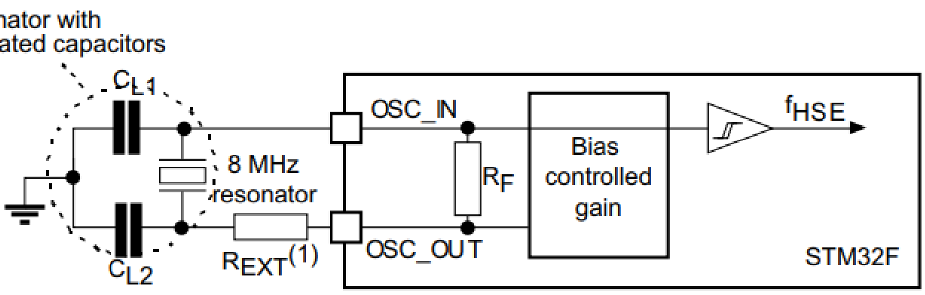


Fig. 36 HSE Oscillator

Note that adding R_{EXT} is not necessary, but adding it is a good practice, as it will limit noise and additional harmonics and distortion. For calculating the same, we'll need to see the crystal's datasheet, as well as another application note from STMicroelectronics, Oscillator design guide for STM8AF/AL/S, STM32 MCUs and MPUs from the site we earlier visited.

The formula for calculating R_{EXT} is

$$g_{mcrit} = 4 \text{ x } ESR \text{ x } (2 \pi F)^2 \text{ x } (C_0 + C_1)^2$$

where,

ESR is equivalent series resistance or R_{EXT}

C_0 is the crystal shunt resistance

C_1 is the normal load capacitance

F is crystal nominal oscillation frequency

To start the oscillation, the primary condition is that $gain_{margin} > 5$, and $gain_{margin}$ can be calculated by

$$gain_{margin} = g_m / g_{mcrit}$$

The C_0 value according to the datasheet is 7pF, load capacitance C_1 is 20pF, frequency is 26MHz, ESR can be calculated now with required g_{mcrit}. But this specific oscillator datasheet has given a table for recommended ESR for different frequencies. Here the ESR value which we'll be using is 40 Ω.

Now let's create the schematic for Boot Modes and NRST (Reset Pin).

According to the datasheet, NRST pin is connected to a permanent pull-up resistor, and in order to reset the board, we need to ground the NRST pin.

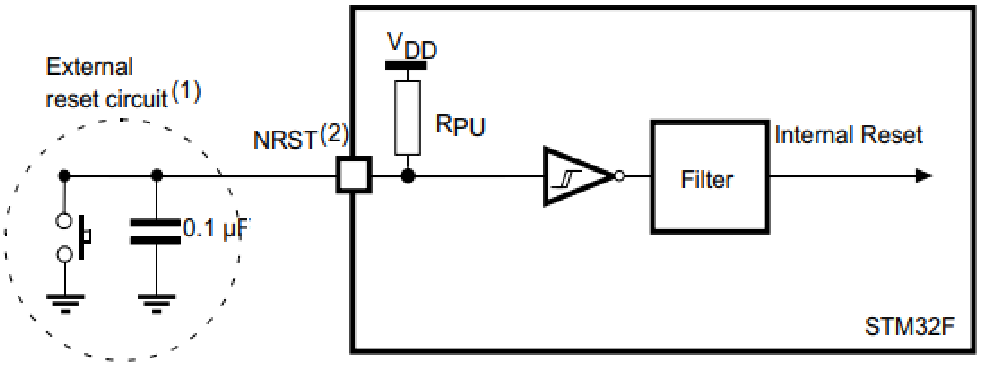


Fig. 37 Recommended NRST pin Protection

Remember to connect the 0.1µF capacitor! Moving on to the boot, there are three differential boot modes according to the datasheet which can be selected, and below is the schematic for the same.

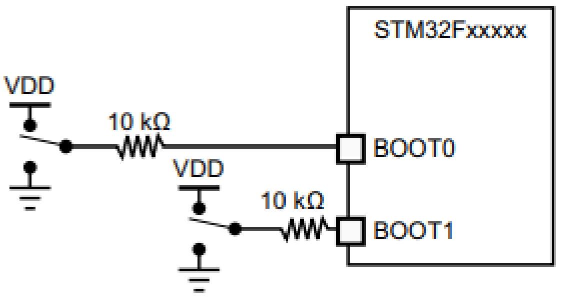


Fig. 38 Boot mode selection example

Moving further, let's add the decoupling capacitors, note that V_{DD} is 3v3 and V_{SS} is GND, and V_{DDA} is for independent power supply for conversion accuracy. By referencing the datasheet, it can be seen that each power supply pairs (V_{SS} and V_{DD}) should be paired with filtering ceramic capacitors of value 100nF and one single tantalum or Ceramic capacitor typically of value 10µF (minimum value can be 4.7µF) and for V_{DDA} and V_{SSA} pairs, we'll be using 10nF.

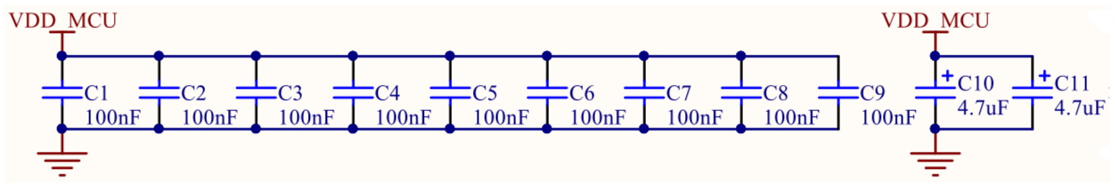


Fig. 39 Decoupling Capacitors

Lastly, we'll be adding the micro-USB FS. Note that USB can be used in several ways i.e., to power the STM board, or to create a virtual com port and use it for debugging. But we can't supply 5v directly to the STM32 board, we need to add a voltage regulator in between, here I'm using AMS1117-3v3 (any other voltage regulator will do). We'll add a USBLC6 ESD (Electrostatic Discharge) protection. We'll need to check the datasheet of AMS1117-3v3, and do the connection accordingly.

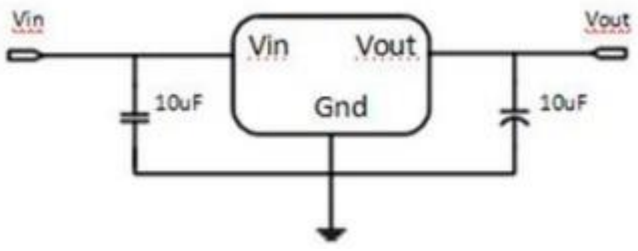


Fig. 40 Application of AMS1117 3v3

And D+ and D- will be connected from USB to USBLC6 and from there directly to the IC. Note that we need a pull-up resistor on the DP pin, this MCU already has one internally, and is not required (Refer the datasheet for your IC).

4 Communication Protocols

STM32 supports various communication protocols. In this chapter we'll see a short overview for them, as the HAL API encompasses all the transmit and receive function for all the communication protocols.

Most of the STM32 supports I^2C, SPI, UART, USART, USB-OTG, Ethernet, CAN, FSMC protocols. One of the great things about HAL API is it provides IT (Interrupt) and DMA (Direct Memory Access) with the communication protocol functions, so we can create Interrupt / DMA by few clicks in the STM32CubeIDE.

The main protocols we'll be looking in this chapter are UART, I^2C and SPI, also note that we've already covered USB-OTG-FS in "Debugging your STM32" section of Chapter 2. In order to cover CAN, we need CAN Transceiver (e.g., MCP2551). Also, in order to keep this book short, I'll be sharing the ideas and all the examples will be covered in a GitHub repository.

UART

Universal asynchronous receiver-transmitter or UART is very easy to implement in STM32.

Just create a new project, and in the Connectivity tab, set any UART to Asynchronous mode (if you are trying it on a Blue Pill, set the USART to Asynchronous mode, also don't forget to do the necessary configurations in the Pinout & Clock section of STM32CubeIDE). Proceed to generate the code.

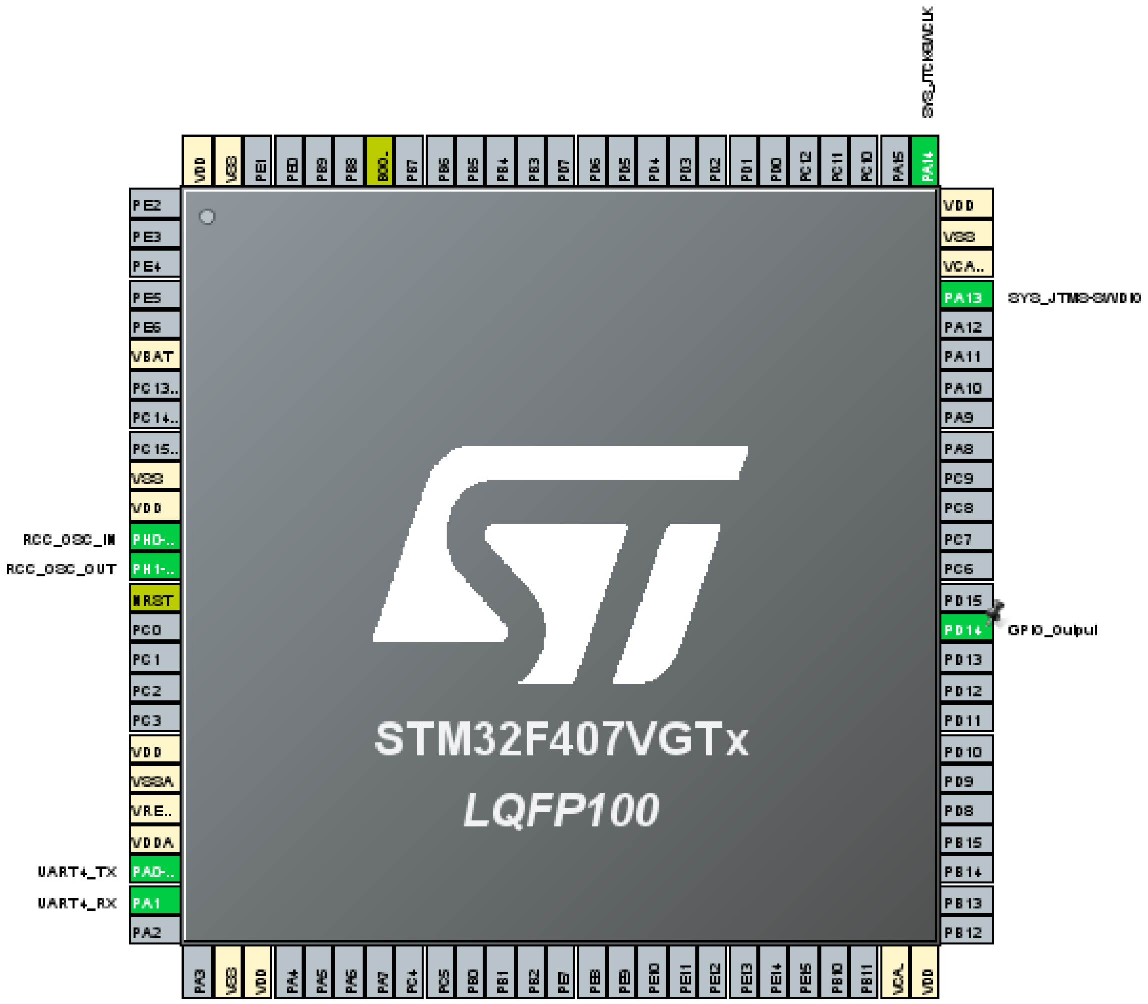


Fig. 40 UART Pin Config

Now before we transmit or receive data, we’ll create a char buffer of desired size in /* User code begin 1 */, and whenever we wish to transmit or receive, we’ll need to pass the variable with type-casting (it needs to be casted to uint8_t * as the function excepts a pointer to uint8_t as an argument), we also need to pass the size we are transmitting, for the same we can use strlen() function.

Below is the code for transmitting a string using UART, if we need to receive, we'll use HAL_UART_Receive(address of UART, data to pass, size of data, and timeout);

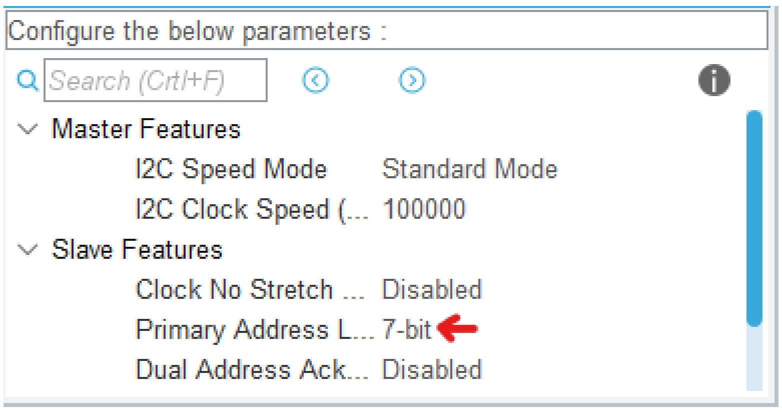
```
/* USER CODE BEGIN 1 */
    char buf[15] = "Hello, World\r\n";
/* USER CODE END 1 */

/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART1_UART_Init();
MX_USB_DEVICE_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
      HAL_UART_Transmit(&huart1, (uint8_t *) buf, strlen(buf), 100);
}
```

UART communication is useful, and some of the sensors work on UART protocol, for example TF-Mini. You may find all the required information in the datasheet of the TF-Mini if you wish to interface it with STM32.

If you want more information regarding UART or any other communication protocol, go through User Manual – HAL and Low layer drivers.

I^{2}C

Inter-Integrated Circuit or I^{2}C is again very easy to implement in STM32CubeIDE, thanks to HAL. Configure the required pins and enable an I^{2}C channel, but here you'll have to note some things.



Configure the below parameters :

Search (Crtl+F)

Master Features

I2C Speed Mode	Standard Mode
I2C Clock Speed (...	100000

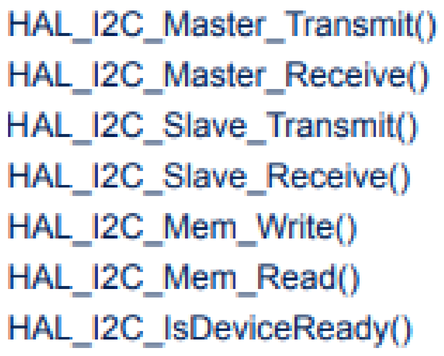
Slave Features

Clock No Stretch ...	Disabled
Primary Address L...	7-bit
Dual Address Ack...	Disabled

Fig. 41 Addressing in I^{2}C

Notice that 7-bit! So, whenever you are passing an 8-bit address, you'll have to left-shift (<<) the address once. If you are interfacing an MPU-6050 with STM32, and using the I^{2}C for the same, if you pass 0x68 as the address, your MPU won't be detected, you'll have to pass 0xD0 (0x68 << 1 = 0xD0).

You can generate the code. And you can try various functions which are given in Fig. 42. Try to create an I^{2}C scanner using IsDeviceReady!

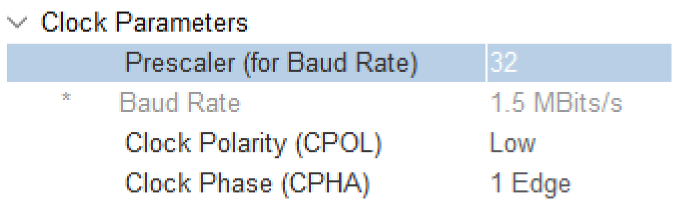


HAL_I2C_Master_Transmit()
HAL_I2C_Master_Receive()
HAL_I2C_Slave_Transmit()
HAL_I2C_Slave_Receive()
HAL_I2C_Mem_Write()
HAL_I2C_Mem_Read()
HAL_I2C_IsDeviceReady()

Fig. 42 Various I^2C functions

SPI

Again, create a new project, and configure the necessary pins. In the Connectivity tab, set the SPI to required settings, like Full-Duplex Slave/Master or anything else from the drop-down list. Here we need to configure the baud rate, initially it's set to 12 Mb/s, configure the Prescaler for baud rate and set the transfer speed as required, here I'll set it to 32 so that my speed goes down to 1.5Mb/s.



Clock Parameters

Prescaler (for Baud Rate)	32
* Baud Rate	1.5 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

Fig. 43 Configuring the Baud Rates

Now in the /* USER CODE BEGIN 2 */, set the SCK pin HIGH, and now whenever we need to send the data, we'll set the SCK pin to LOW state, transmit / receive the data, and again set the SCK pin to high. To transmit and receive the data, we'll use

HAL_SPI_Transmit() && HAL_SPI_Receive().

That's all for the SPI section, I've tried to convey the idea here, but note that I'll cover this topic in depth, and upload the well documented readme and project in GitHub. By the way I already have created some necessary libraries and uploaded them on GitHub, Happy Learning!

Now what?

If you have completed this book, you should practice more and more. There are lots of things which I've not covered in this book, there are some great resources on YouTube as well as on the web, try to go through them and try them as well.

Some of the resources are:

Digi-Key YouTube

Mutex Embedded - Education - YouTube

DeepBlue

Appendix

Programming STM32 using Arduino IDE

Programming STM32 using MBED

STM32 PCB Design